Faculty of Sciences of the University
of Lisbon

Advanced Machine Learning
Assignment 1 - Backpropagation

55761 · AntónioRebelo
55877 · MarcoMercier

taught by
Professor Luis Correia

February 15, 2021

1 Problem 1

a) The equation for the local field of the first neuron (v_1) is:
$v_1 = w_{11}x_1 + w_{12}x_2 + b_1 = x_1 + x_2 - 1.5$

For the second neuron (v_2):
$v_2 = w_{21}x_1 + w_{22}x_2 + w_{23}\phi(v_1) + b_2 = x_1 + x_2 - 2\phi(v_1) - 0.5$

Thus, the output will be:
$O = \phi(x_1 + x_2 - 2\phi(x_1 + x_2 - 1.5) - 0.5)$

Where:

$$\phi(n) = \begin{cases} 0 & n \leq 0 \\ 1 & n > 0 \end{cases}$$

is the activation function. The separation planes will be where $n = 0$, which is when the activation function makes the neuron fire differently.

$v_1 = 0 \Leftrightarrow x_1 = 1.5 - x_2$
$v_2 = 0 \Leftrightarrow x_1 = 0.5 + 2\phi(v_1) - x_2$

Since $x_1, x_2 \in \{0, 1\}$, lets explore for both values:

$$\mathrm{x}_1 = 0 \Rightarrow \begin{cases} N_1 & x_2 = 1.5 \\ N_2 & x_2 = 0.5 \end{cases}$$
$$\mathrm{x}_2 = 0 \Rightarrow \begin{cases} N_1 & x_1 = 1.5 \\ N_2 & x_1 = 0.5 \end{cases}$$

Thus, the separation planes is:

$$\begin{cases} x_2 = -x_1 + 1.5 \\ x_2 = -x_1 + 0.5 \end{cases}$$

b) The Truth table is obtained by using the output equation above:

x_0	x_1	Output
0	0	0
0	1	1
1	0	1
1	1	0

c) Considering $a = 0$ for the activation function $\phi(v) = \sigma(v) = \frac{1}{1+e^{-v}}$, learning rate $\eta = 1$, the initial weights to be the ones used before and that the previous Truth Table provides the desired outputs.

The delta rule is $\Delta w_{ji} = \eta \delta_j y_i$, where δ_j is the local gradient of neuron j and y_i the output of the neuron i. Using the sigmoid activation function, the local gradient of an output neuron k is:

$\delta_k = y_k(1 - y_k)(d_k - y_k)$

and the local gradient for a hidden layer neuron j is:

$\delta_j = y_j(1 - y_j) \sum_k \delta_k w_{kj}$

Lets run the first iteration of the first epoch for $(x_1, x_2, O) = (0, 0, 0)$:

$v_1 = -1.5 \Rightarrow y_1 = \phi(v_1) = \sigma(v_1) = 0.182$

$v_2 = -0.5 - 2y_1 = -0.864 \Rightarrow y_2 = 0.297$

$\delta_2 = 0.297(1 - 0.297)(0 - 0.297) = -0.062$

Lets adjust the weights of the output neuron:

$w_{21} = w_{21} + \eta \delta_2 x_1 = 1 + 1 * -0.062 * 0 = 1$

$w_{22} = w_{22} + \eta \delta_2 x_2 = 1 + 0 = 1$

$w_{23} = w_{23} + \eta \delta_2 y_1 = -2 + 1 * -0.062 * 0.182 = -2.011$

$b_2 = b_2 + \eta \delta_2 = -0.5 + 1 * -0.062 = -0.562$

Now for the hidden layer neuron:

$\delta_1 = y_1(1 - y_1)\delta_2 w_{23} = 0.182(1 - 0.182) * -0.062 * -2 = 0.0185$

$w_{11} = w_{11} + \eta \delta_1 x_1 = 1 + 1 * 0.0185 * 0 = 1$

$w_{12} = w_{12} + \eta \delta_1 x_2 = 1 + 1 * 0.0185 * 0 = 1$

$b_1 = b_1 + \eta \delta_1 = -1.5 + 1 * 0.0185 = -1.482$

Hand-in-1_ex3

February 14, 2021

[1]:
```
import torch.nn as nn
import torch
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np
import random
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

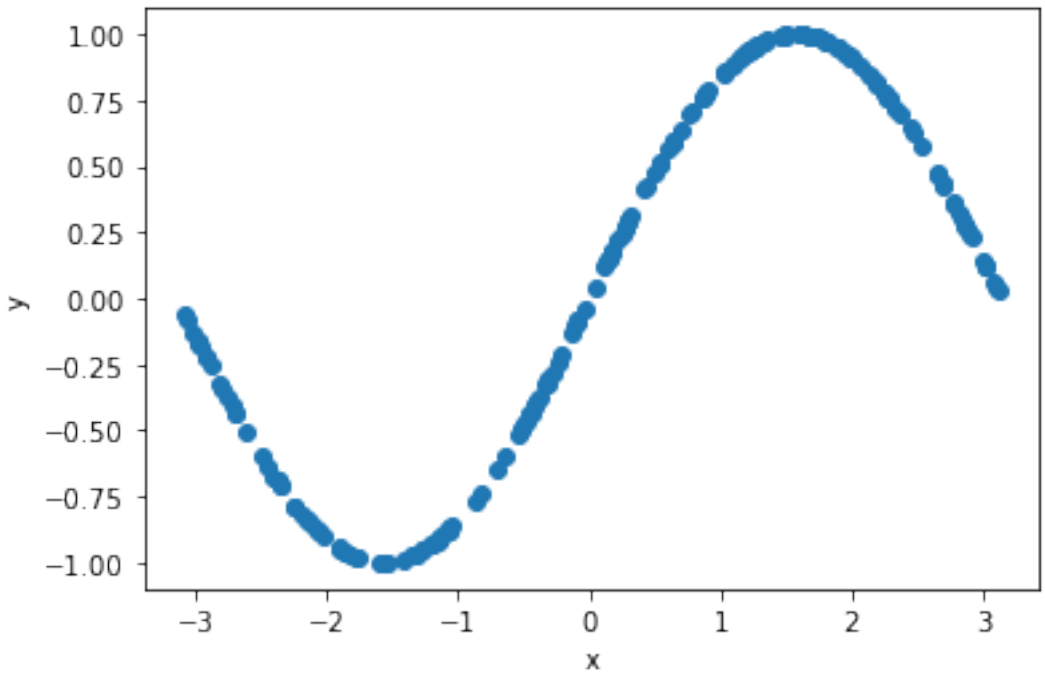
We start by definning x and y as is requested

[2]:
```
#to guarantee reproducible results
np.random.seed(0)

x = [random.uniform(-np.pi,np.pi) for _ in range(200)]
y = np.sin(x)
```

[3]:
```
#Visualizing the data:
fig, ax = plt.subplots()
ax.scatter(x,y);
ax.set_xlabel('x')
ax.set_ylabel('y')
```

[3]:
```
Text(0, 0.5, 'y')
```



[4]:
```
#pytorch only trains on its special Variable class so we need to convert the␣
 ↪variables
x = Variable(torch.tensor(x))
y = Variable(np.sin(x))
```

[5]:
```
#the dimensions of the vectors also need to be adjusted because of how pytorch␣
 ↪multiplies matrices to train NN's
x = x.unsqueeze(dim=-1)
y = y.unsqueeze(dim=-1)
```

We split the data into trainning, test and validation set

[6]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
 ↪15,random_state = 20)
```

[7]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.
 ↪17, random_state = 20)
```

0.1 Setting up the network

[8]:
```
#set the random seed to initialize weights
torch.manual_seed(1);
```

[9]:
```
#Here comes the network with one hidden layer of two neurons (nn.Linear(1,2))␣
↪and a tanh activation function
class NN(torch.nn.Module):
    def __init__(self):
        super(NN,self).__init__()
        self.layer1 = nn.Linear(1,15)
        self.predict = nn.Linear(15,1)

    def forward(self,x):
        x = torch.tanh(self.layer1(x))
        x = self.predict(x)
        return x
```

0.2 Trainning the model

[10]:
```
#instantiate the network
Net = NN()
#set up the optimizer as Adam with learning rate 0.2
opt = torch.optim.Adam(Net.parameters(), lr = 0.2)
#set up the los function as MSE
loss_func = torch.nn.MSELoss()
```

[11]:
```
#now the actual training
#we set the number of trainning epochs
n_epochs = 300


for epoch in range(n_epochs):

    #calculate predictions
    preds = Net(x_train)

    #calculate loss
    loss = loss_func(preds,y_train)

    #set gradients to zero
    opt.zero_grad()
    #backpropagate from the loss
    loss.backward()
    #update weights
    opt.step()
```

0.3 Results on validation set

Note: in a real world problem, this would be overfitting, but in a academic setting such as this one, that has no meaning.

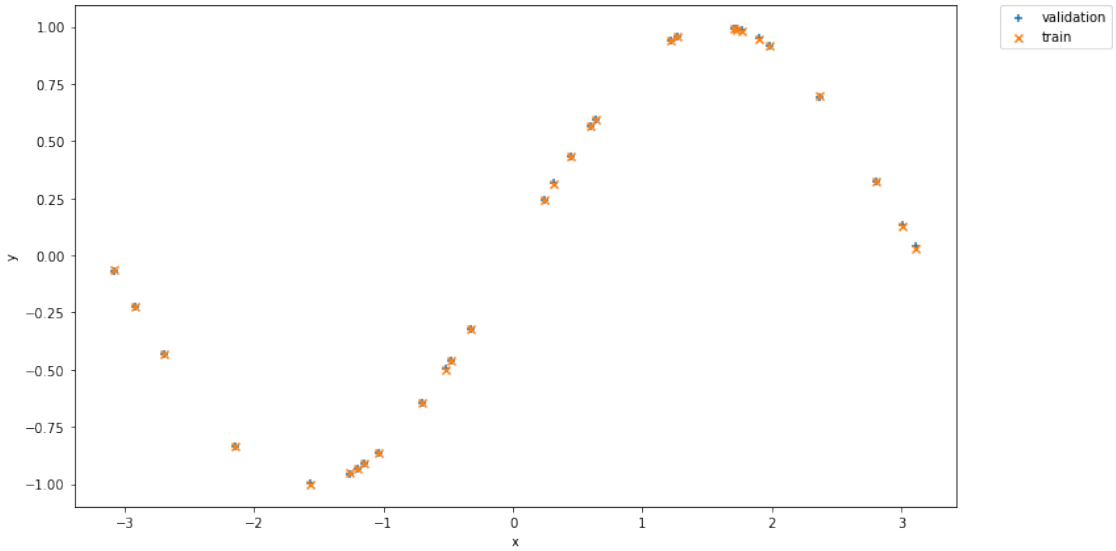
[12]:
```
#the predictions for the validation set:
preds_val = Net(x_val)

#the MSE
loss_func(preds_val,y_val)
```

[12]:
```
tensor(1.0091e-05, grad_fn=<MseLossBackward>)
```

The loss is very very low so the model is fitting very well. Checking the results graphically:

[13]:
```
fig, ax = plt.subplots(figsize=(12,7))
ax.scatter(x_val.flatten().detach().numpy(),preds_val.flatten().detach().
 ↪numpy(), label = "validation",marker="+");
ax.scatter(x_val.flatten().detach().numpy(),y_val.flatten().detach().numpy() ,␣
 ↪label = "train",marker="x");
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.);
```



[ ]:
```

```

0.4 Results on test set

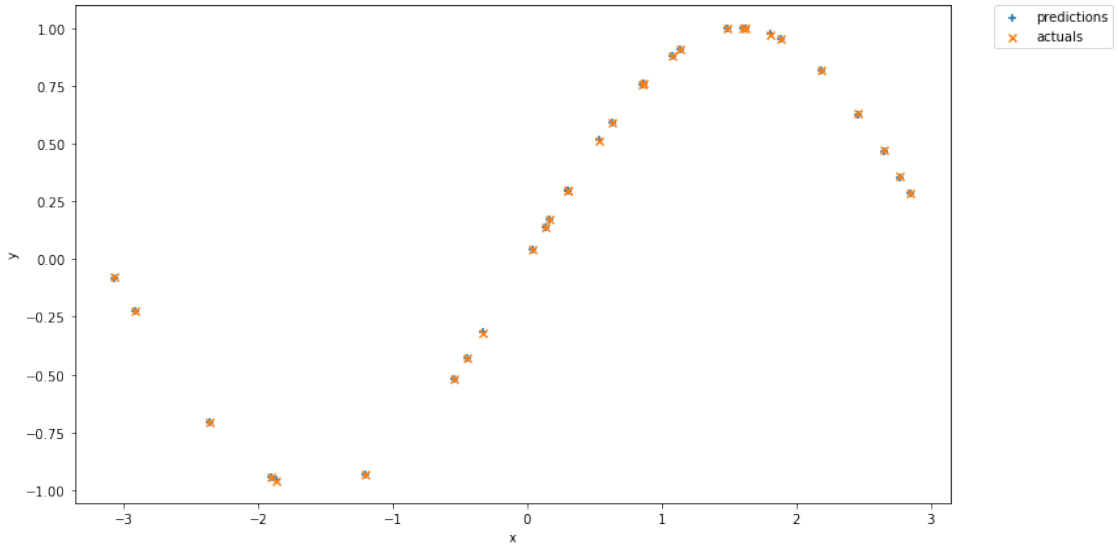
[14]:
```
#the predictions for the test set:
preds = Net(x_test)

#the MSE
loss_func(preds,y_test)
```

[14]:
```
tensor(5.2821e-06, grad_fn=<MseLossBackward>)
```

The loss is very very low so the model is fitting very well. Checking the results graphically:

[15]:
```
fig, ax = plt.subplots(figsize=(12,7))
ax.scatter(x_test.flatten().detach().numpy(),preds.flatten().detach().numpy(),␣
 ↪label = "predictions",marker="+");
ax.scatter(x_test.flatten().detach().numpy(),y_test.flatten().detach().numpy()␣
 ↪, label = "actuals",marker="x");
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.);
```



[ ]:

[ ]:

[ ]: