Inicializando con pesos identicos

Objetivos

Definir una red neuronal con los pesos inicializados con el mismo valor y ver que pasa

Tabla de contenido

Veremos el problema de inicializar los pesos con el mismo valor.

Preparación

In [1]:
```
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

In [2]:
```
# Import the libraries we need for this lab

import torch
import torch.nn as nn
from torch import sigmoid
import matplotlib.pylab as plt
import numpy as np
torch.manual_seed(0)
```

Out[2]:
```
<torch._C.Generator at 0x248ce2a7250>
```

Para graficar el modelo:

In [3]:
```
# The function for plotting the model

def PlotStuff(X, Y, model, epoch, leg=True):

    plt.plot(X.numpy(), model(X).detach().numpy(), label=('epoch ' + str(epoch)))
    plt.plot(X.numpy(), Y.numpy(), 'r')
    plt.xlabel('x')
    if leg == True:
        plt.legend()
    else:
        pass
```

Módulo red neuronal y función de entrenamiento

Definimos las activaciones y la salida de la primera capa lineal como un atributo; esto NO es una buena práctica.

In [4]:
```
# Define the class Net

class Net(nn.Module):

    # Constructor
    def __init__(self, D_in, H, D_out):
        super(Net, self).__init__()
        # hidden layer
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)
        # Define the first linear layer as an attribute, this is not good practice
        self.a1 = None
        self.l1 = None
        self.l2=None

    # Prediction
    def forward(self, x):
        self.l1 = self.linear1(x)
        self.a1 = sigmoid(self.l1)
        self.l2=self.linear2(self.a1)
        yhat = sigmoid(self.linear2(self.a1))
        return yhat
```

Definimos la función de entrenamiento:

In [5]:
```
# Define the training function

def train(Y, X, model, optimizer, criterion, epochs=1000):
    cost = []
    total=0
    for epoch in range(epochs):
        total=0
        for y, x in zip(Y, X):
            yhat = model(x)
            loss = criterion(yhat, y)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            #cumulative loss
            total+=loss.item()
        cost.append(total)
        if epoch % 300 == 0:
            PlotStuff(X, Y, model, epoch, leg=True)
            plt.show()
            model(X)
            plt.scatter(model.a1.detach().numpy()[:, 0], model.a1.detach().numpy()[:,
            plt.title('activations')
            plt.show()
    return cost
```

Creamos algunos datos

In [6]:
```
# Make some data

X = torch.arange(-20, 20, 1).view(-1, 1).type(torch.FloatTensor)
Y = torch.zeros(X.shape[0])
Y[(X[:, 0] > -4) & (X[:, 0] < 4)] = 1.0
```

Definir la red neuronal, inicializar los pesos con el mismo valor, definir costo, optimizador y entrenar el modelo

Creamos la función de pérdida de entropía cruzada

In [7]:
```
# The loss function

def criterion_cross(outputs, labels):
    out = -1 * torch.mean(labels * torch.log(outputs) + (1 - labels) * torch.log(1 - o
    return out
```

Definimos la red neuronal:

In [8]:
```
# Train the model
# size of input
D_in = 1
# size of hidden layer
H = 2
# number of outputs
D_out = 1
# learning rate
learning_rate = 0.1
# create the model
model = Net(D_in, H, D_out)
```

La siguiente es la inicialización por defecto de PyTorch:

In [9]:
```
model.state_dict()
```

Out[9]:
```
OrderedDict([('linear1.weight',
              tensor([[-0.0075],
                      [ 0.5364]])),
             ('linear1.bias', tensor([-0.8230, -0.7359])),
             ('linear2.weight', tensor([[-0.2723,  0.1896]])),
             ('linear2.bias', tensor([-0.0140]))])
```

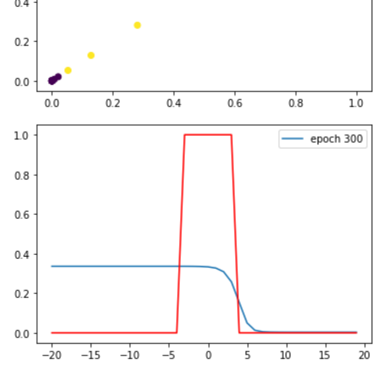
Inicializamos todos los pesos en 1 y los sesgos en 0:

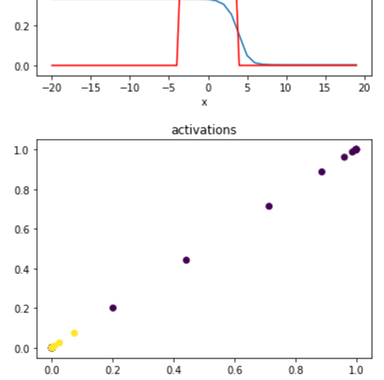
In [10]:
```
model.state_dict()['linear1.weight'][0]=1.0
model.state_dict()['linear1.weight'][1]=1.0
model.state_dict()['linear1.bias'][0]=0.0
model.state_dict()['linear1.bias'][1]=0.0
model.state_dict()['linear2.weight'][0]=1.0
model.state_dict()['linear2.bias'][0]=0.0
model.state_dict()
```

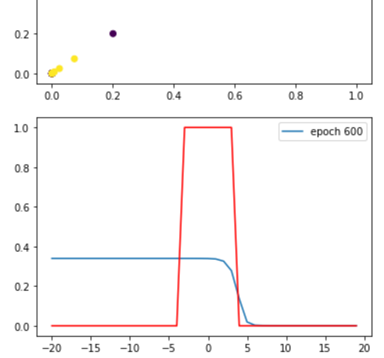
Out[10]:
```
OrderedDict([('linear1.weight',
              tensor([[1.],
                      [1.]])),
             ('linear1.bias', tensor([0., 0.])),
             ('linear2.weight', tensor([[1., 1.]])),
             ('linear2.bias', tensor([0.]))])
```

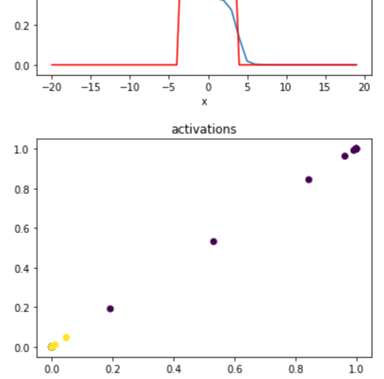
Optimizador y entrenamiento del modelo:

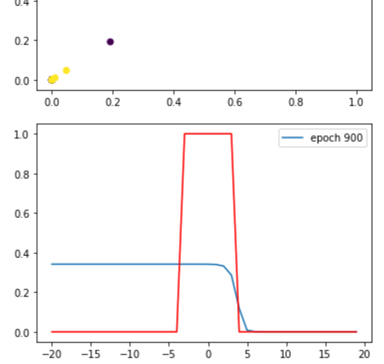
In [11]:
```
#optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
#train the model usein
cost_cross = train(Y, X, model, optimizer, criterion_cross, epochs=1000)
#plot the loss
plt.plot(cost_cross)
plt.xlabel('epoch')
plt.title('cross entropy loss')
```

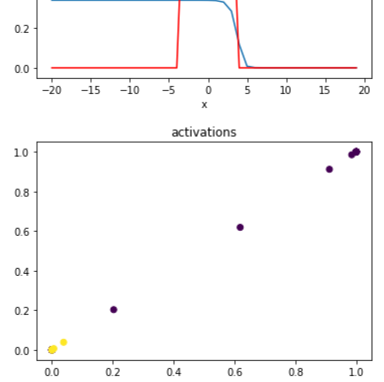


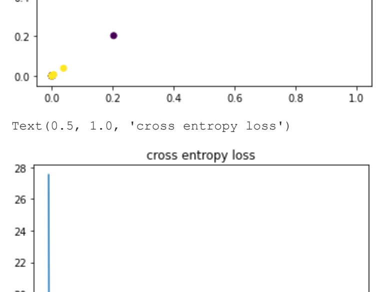


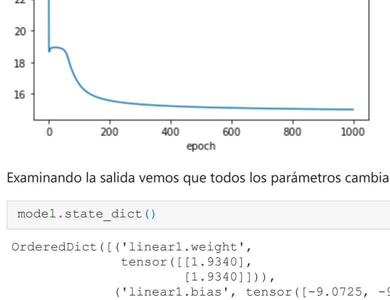




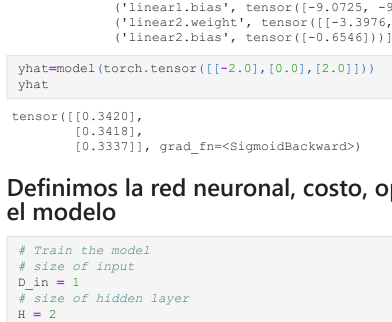








Out[11]:
```
Text(0.5, 1.0, 'cross entropy loss')
```



Examinando la salida vemos que todos los parámetros cambiaron de la misma forma:

In [12]:
```
model.state_dict()
```

Out[12]:
```
OrderedDict([('linear1.weight',
              tensor([[1.9340],
                      [1.9340]])),
             ('linear1.bias', tensor([-9.0725, -9.0725])),
             ('linear2.weight', tensor([[-3.3976, -3.3976]])),
             ('linear2.bias', tensor([-0.6546]))])
```

In [13]:
```
yhat=model(torch.tensor([[-2.0],[0.0],[2.0]]))
yhat
```

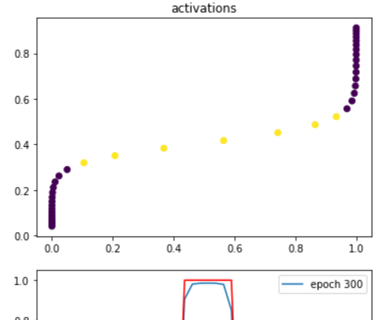
Out[13]:
```
tensor([[0.3420],
        [0.3418],
        [0.3337]], grad_fn=<SigmoidBackward>)
```

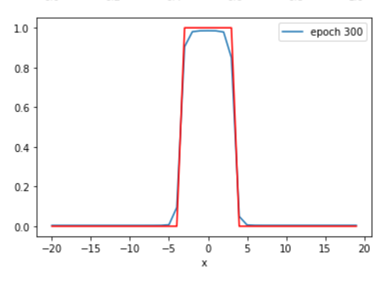
Definimos la red neuronal, costo, optimizador y entrenamos el modelo

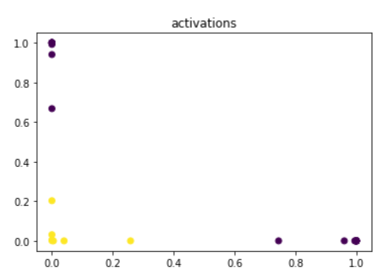
In [14]:
```
# Train the model
# size of input
D_in = 1
# size of hidden layer
H = 2
# number of outputs
D_out = 1
# learning rate
learning_rate = 0.1
# create the model
model = Net(D_in, H, D_out)
```

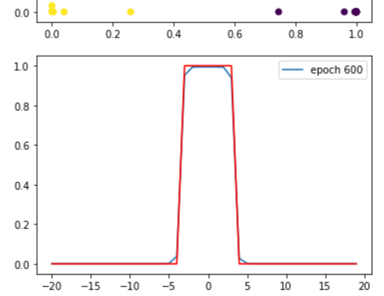
Repetimos los pasos previos pero ahora usando MSE:

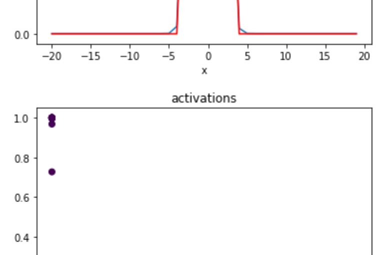
In [15]:
```
#optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
#train the model usein
cost_cross = train(Y, X, model, optimizer, criterion_cross, epochs=1000)
#plot the loss
plt.plot(cost_cross)
plt.xlabel('epoch')
plt.title('cross entropy loss')
```

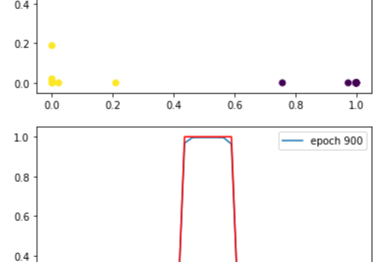


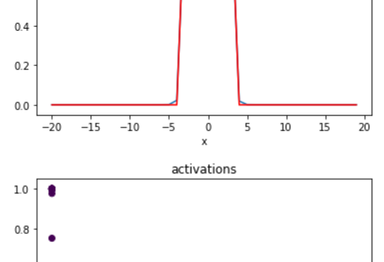


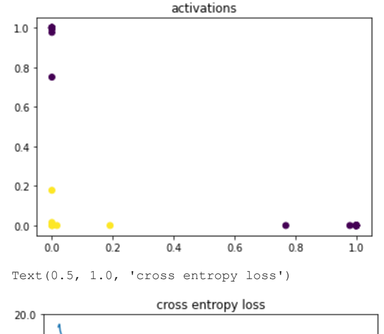












Out[15]:
```
Text(0.5, 1.0, 'cross entropy loss')
```