CLASSIFICATION

Module 3

v-cardona

Deep Learning with Python and PyTorch

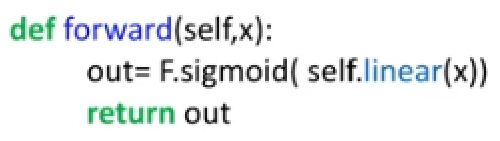
Table of contents

Logistic regression prediction

It classificates into 0 or 1

- Create a logistic regression object with the `nn.Sequential` model with a one-dimensional input:
  model = nn.Sequential(nn.Linear(1, 1), nn.Sigmoid())

The only difference between linear regression is you add the sigmoid function and the forward function.



```
def forward(self,x):
        out= F.sigmoid( self.linear(x))
        return out
```

Training logistic regression

The initial value it is important because it can make that the model do not converge. I you use Mean Square Error and you select a bad initialization value for your parameter the cost or total loss surface will be flat and the parameter will not update.

Bernoulli distribution and maximum likelihood estimation

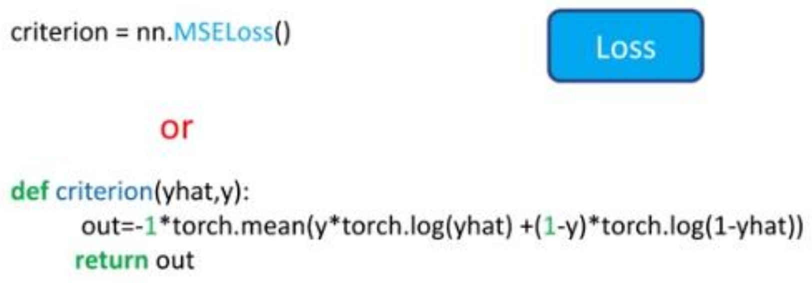
For each of the individual probabilities is a function multiplied together. The likelihood of the function is represented by the overlapping values and the goal is to find the value of the parameter that maximizes this function. It turns out that it is simple to maximize the log of the likelihood function given by the following expression. Because the log is monotonically, increasing it may change the shape of the function but has not changed the location of a maximum value for the parameter theta.

Cross entropy loss

The advantage of the cross-entropy loss over the Mean Square Error is that there are contours all over the cost or total loss surface; as a result, the algorithm will converge to a local minimum.

- Expression for the cross entropy loss:
  `out=-1*torch.mean(y*torch.log(yhat) +(1-y)*torch.log(1-yhat))`

- Calculate the loss:
  Mean squared error -> MSE
  Binary cross entropy -> BCE or function criterion() bellow



```
criterion = nn.MSELoss()

or

def criterion(yhat,y):
        out=-1*torch.mean(y*torch.log(yhat) +(1-y)*torch.log(1-yhat))
        return out
```

Softmax regression

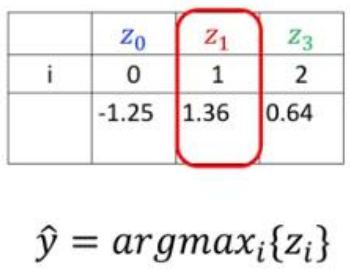
When we create the SoftMax class or custom module, we will vary the parameter out_size. This will correspond to the number of classes. We will obtain the values that the sample correspond to each class and if we apply a max function to it, we will get to which class corresponds.

- *Max_value, yhat = z.max(1)*

Yhat is a row where each column is the index of the maximum of one sample data

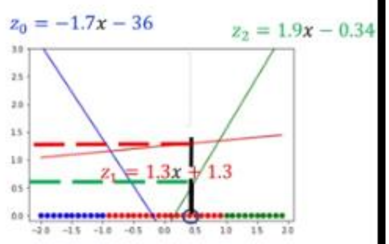
Softmax Function

Each class will perform a linear separation and the maximum will chose at which class belog



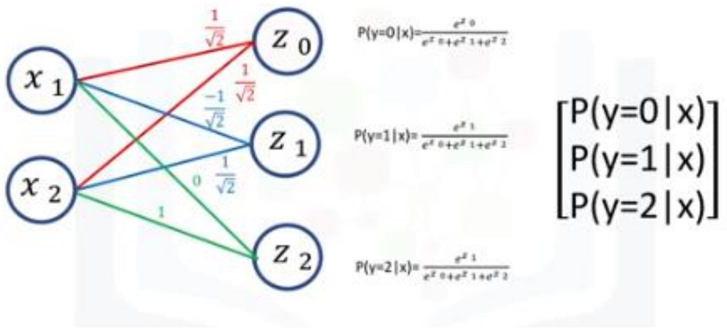
	z_0	z_1	z_3
i	0	1	2
	-1.25	1.36	0.64

$$\hat{y} = argmax_i\{z_i\}$$



Instead of lines use vectors as a classificatory and each sample is a vector too, so the data will be classified to the class which vector is closer.

Just to note, the reason they call it the SoftMax function is the actual distances are converted into probabilities using the following formula or simply normalize the function. This behaves similar to the threshold function, it is kind of a soft approximation, and it gives you a much smoother surface.



x_1
x_2
z_0
z_1
z_2
$\frac{1}{\sqrt{2}}$
$\frac{-1}{\sqrt{2}}$
0
1
$P(y=0|x)=\frac{e^{z_0}}{e^{z_0}+e^{z_1}+e^{z_2}}$
$P(y=1|x)=\frac{e^{z_1}}{e^{z_0}+e^{z_1}+e^{z_2}}$
$P(y=2|x)=\frac{e^{z_2}}{e^{z_0}+e^{z_1}+e^{z_2}}$

$$\begin{bmatrix} P(y=0|x) \\ P(y=1|x) \\ P(y=2|x) \end{bmatrix}$$

- criterion = nn.CrossEntropyLoss()

- Train model:

```
n_epochs = 10
loss_list = []
accuracy_list = []
N_test = len(validation_dataset)

def train_model(n_epochs):
  for epoch in range(n_epochs):
    for x, y in train_loader:
      optimizer.zero_grad()
      z = model(x.view(-1, 28 * 28))
      loss = criterion(z, y)
      loss.backward()
      optimizer.step()

    correct = 0
    # perform a prediction on the validationdata
    for x_test, y_test in validation_loader:
      z = model(x_test.view(-1, 28 * 28))
      _, yhat = torch.max(z.data, 1)
      correct += (yhat == y_test).sum().item()
    accuracy = correct / N_test
    loss_list.append(loss.data)
    accuracy_list.append(accuracy)

train_model(n_epochs)
```