Redes neuronales con momento

Objectivos

Entrenar varias redes neuronales con diferentes momentos

Comparar los resultados

Tabla de contenido

Veremos cómo los diferentes valores del momento afectan la tasa de convergencia de una red neuronal

- Módulo red neuronal y función de entrenamiento
- Entrenar diferentes redes neuronales con diferentes momentos
- Comparar los resultados

Preparación

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

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

import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from matplotlib.colors import ListedColormap
from torch.utils.data import Dataset, DataLoader

torch.manual_seed(1)
np.random.seed(1)
```

Función para graficar:

In [3]:
```
# Define a function for plot the decision region

def plot_decision_regions_3class(model, data_set):
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA','#00AAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00','#00AAFF'])
    X=data_set.x.numpy()
    y=data_set.y.numpy()
    h = .02
    x_min, x_max = X[:, 0].min() - 0.1 , X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1 , X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
    XX=torch.torch.Tensor(np.c_[xx.ravel(), yy.ravel()])
    _,yhat=torch.max(model(XX),1)
    yhat=yhat.numpy().reshape(xx.shape)
    plt.pcolormesh(xx, yy, yhat, cmap=cmap_light)
    plt.plot(X[y[:]==0,0], X[y[:]==0,1], 'ro', label='y=0')
    plt.plot(X[y[:]==1,0], X[y[:]==1,1], 'go', label='y=1')
    plt.plot(X[y[:]==2,0], X[y[:]==2,1], 'o', label='y=2')
    plt.title("decision region")
    plt.legend()
```

Creamos una clase dataset:

In [4]:
```
# Create the dataset class

class Data(Dataset):

    #  modified from: http://cs231n.github.io/neural-networks-case-study/
    # Constructor
    def __init__(self, K=3, N=500):
        D = 2
        X = np.zeros((N * K, D)) # data matrix (each row = single example)
        y = np.zeros(N * K, dtype='uint8') # class labels
        for j in range(K):
          ix = range(N * j, N * (j + 1))
          r = np.linspace(0.0, 1, N) # radius
          t = np.linspace(j * 4, (j + 1) * 4, N) + np.random.randn(N) * 0.2 # theta
          X[ix] = np.c_[r * np.sin(t), r * np.cos(t)]
          y[ix] = j

        self.y = torch.from_numpy(y).type(torch.LongTensor)
        self.x = torch.from_numpy(X).type(torch.FloatTensor)
        self.len = y.shape[0]

    # Getter
    def __getitem__(self, index):
        return self.x[index], self.y[index]

    # Get Length
    def __len__(self):
        return self.len

    # Plot the diagram
    def plot_data(self):
        plt.plot(self.x[self.y[:] == 0, 0].numpy(), self.x[self.y[:] == 0, 1].numpy(),
        plt.plot(self.x[self.y[:] == 1, 0].numpy(), self.x[self.y[:] == 1, 1].numpy(),
        plt.plot(self.x[self.y[:] == 2, 0].numpy(),self.x[self.y[:] == 2, 1].numpy(),
        plt.legend()
```

Módulo red neuronal y función de entrenamiento

Creamos el módulo red neuronal usando `ModuleList()`

In [5]:
```
# Create dataset object

class Net(nn.Module):

    # Constructor
    def __init__(self, Layers):
        super(Net, self).__init__()
        self.hidden = nn.ModuleList()
        for input_size, output_size in zip(Layers, Layers[1:]):
            self.hidden.append(nn.Linear(input_size, output_size))

    # Prediction
    def forward(self, activation):
        L = len(self.hidden)
        for (l, linear_transform) in zip(range(L), self.hidden):
            if l < L - 1:
                activation = F.relu(linear_transform(activation))
            else:
                activation = linear_transform(activation)
        return activation
```

Creamos la función para entrenar el modelo:

In [6]:
```
# Define the function for training the model

def train(data_set, model, criterion, train_loader, optimizer, epochs=100):
    LOSS = []
    ACC = []
    for epoch in range(epochs):
        for x, y in train_loader:
            optimizer.zero_grad()
            yhat = model(x)
            loss = criterion(yhat, y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        LOSS.append(loss.item())
        ACC.append(accuracy(model,data_set))

    results ={"Loss":LOSS, "Accuracy":ACC}
    fig, ax1 = plt.subplots()
    color = 'tab:red'
    ax1.plot(LOSS,color=color)
    ax1.set_xlabel('epoch', color=color)
    ax1.set_ylabel('total loss', color=color)
    ax1.tick_params(axis = 'y', color=color)

    ax2 = ax1.twinx()
    color = 'tab:blue'
    ax2.set_ylabel('accuracy', color=color)  # we already handled the x-label with ax
    ax2.plot(ACC, color=color)
    ax2.tick_params(axis='y', color=color)
    fig.tight_layout()  # otherwise the right y-label is slightly clipped

    plt.show()
    return results
```

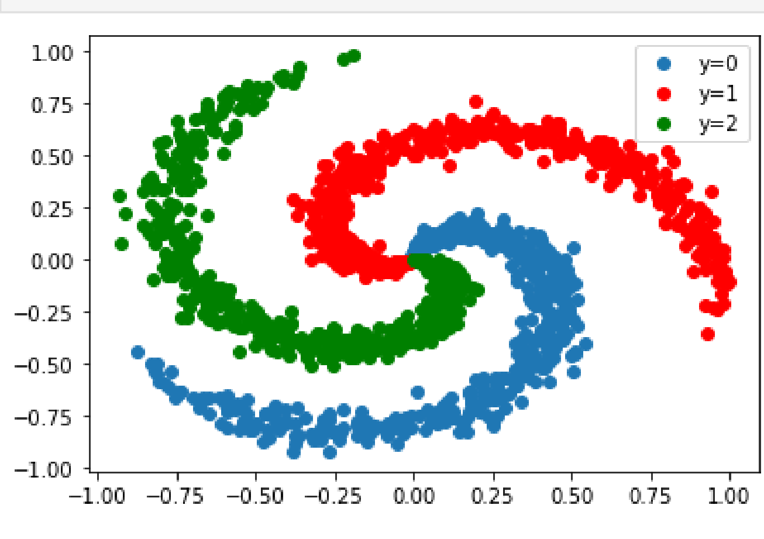
Función para calcular la precisión:

In [7]:
```
# Define a function for calculating accuracy

def accuracy(model, data_set):
    _, yhat = torch.max(model(data_set.x), 1)
    return (yhat == data_set.y).numpy().mean()
```

Entrenamos diferentes redes neuronales con diferentes momentos

Creamos un objeto Dataset usando Data

In [8]:
```
# Create the dataset and plot it

data_set = Data()
data_set.plot_data()
data_set.y = data_set.y.view(-1)
```

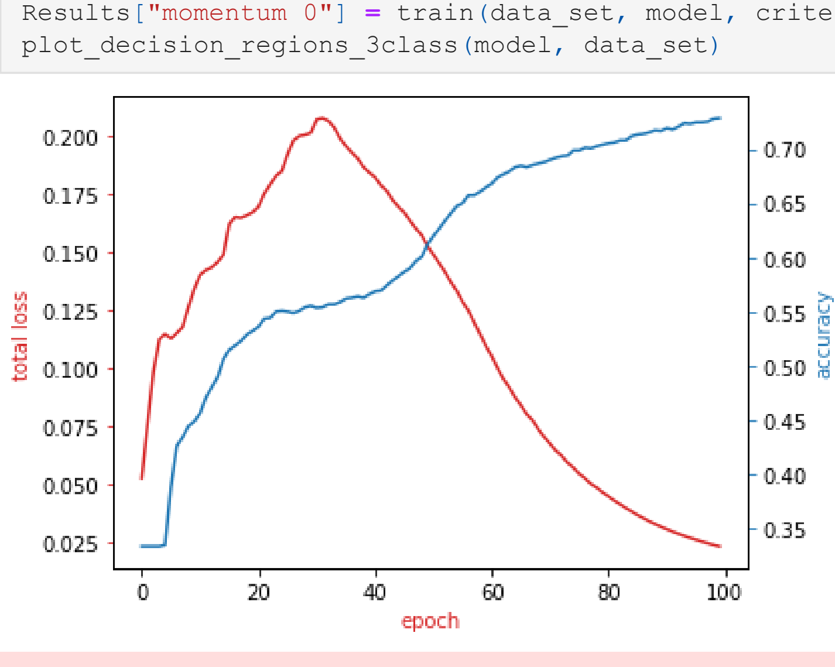


Diccionario que contiene diferentes valores de costo y precisión para cada epoch para diferentes valores del momento.

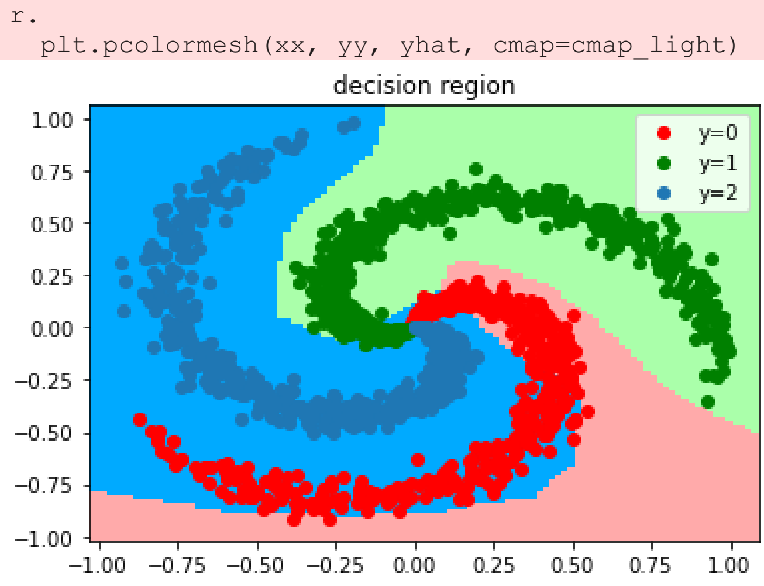
In [9]:
```
# Initialize a dictionary to contain the cost and accuracy

Results = {"momentum 0": {"Loss": 0, "Accuracy:": 0}, "momentum 0.1": {"Loss": 0, "Acc
```

Red para clasificar 3 clases con 1 capa oculta con 50 neuronas y un momento de 0.

In [10]:
```
# Train a model with 1 hidden layer and 50 neurons

Layers = [2, 50, 3]
model = Net(Layers)
learning_rate = 0.10
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
train_loader = DataLoader(dataset=data_set, batch_size=20)
criterion = nn.CrossEntropyLoss()
Results["momentum 0"] = train(data_set, model, criterion, train_loader, optimizer, epo
plot_decision_regions_3class(model, data_set)
```

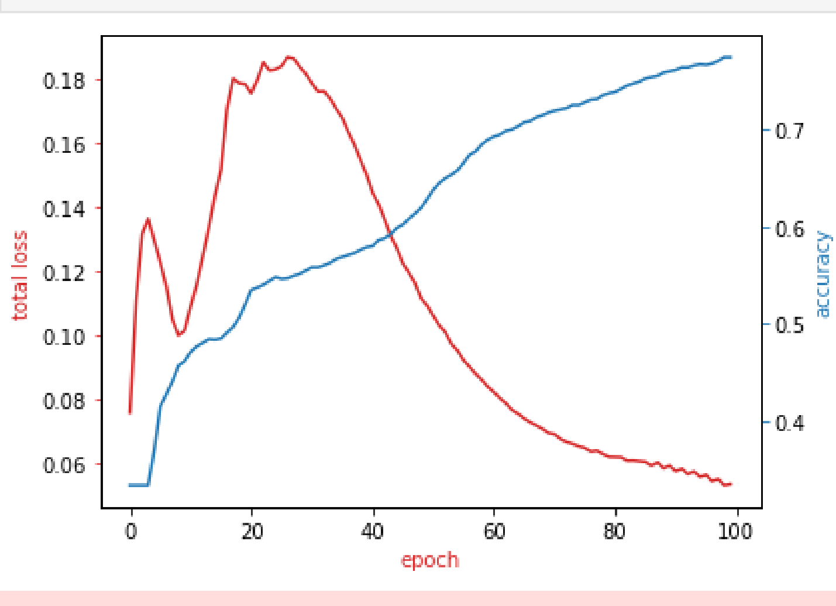


```
<ipython-input-3-f58ec7e51729>:15: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corne
rs of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud',
or set rcParams['pcolor.shading'].  This will become an error two minor releases late
r.
  plt.pcolormesh(xx, yy, yhat, cmap=cmap_light)
```

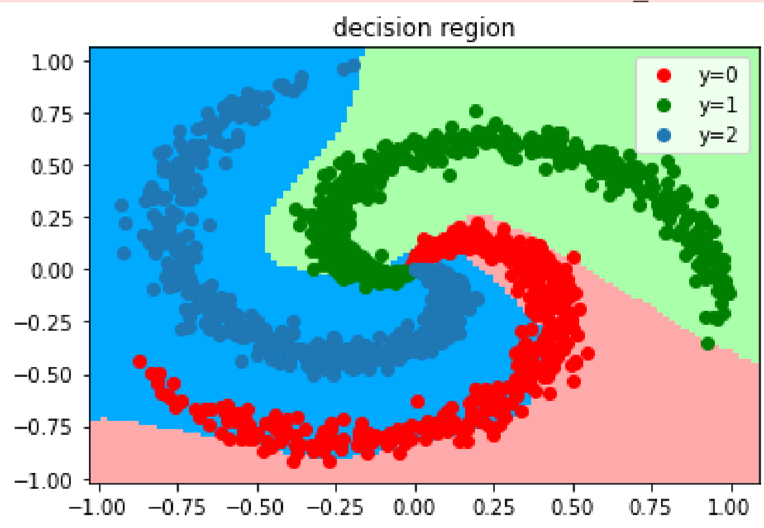


Red para clasificar 3 clases con 1 capa oculta con 50 neuronas y un momento de 0.1

In [11]:
```
# Train a model with 1 hidden layer and 50 neurons with 0.1 momentum

Layers = [2, 50, 3]
model = Net(Layers)
learning_rate = 0.10
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0.1)
train_loader = DataLoader(dataset=data_set, batch_size=20)
criterion = nn.CrossEntropyLoss()
Results["momentum 0.1"] = train(data_set, model, criterion, train_loader, optimizer, e
plot_decision_regions_3class(model, data_set)
```

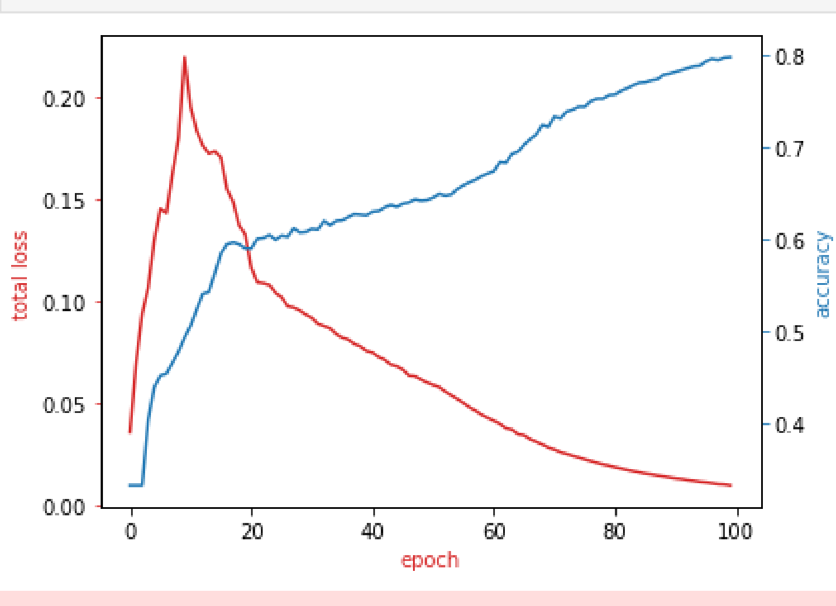


```
<ipython-input-3-f58ec7e51729>:15: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corne
rs of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud',
or set rcParams['pcolor.shading'].  This will become an error two minor releases late
r.
  plt.pcolormesh(xx, yy, yhat, cmap=cmap_light)
```

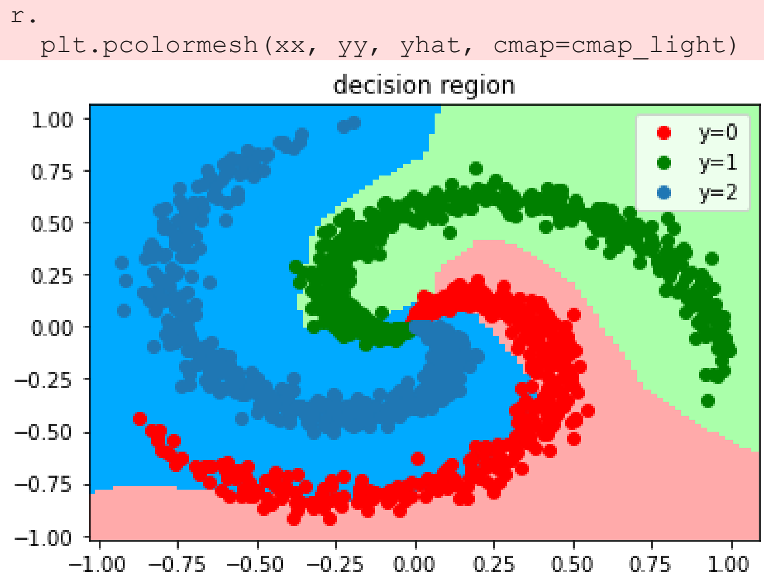


Create a netwRed para clasificar 3 clases con 1 capa oculta con 50 neuronas y un momento de 0.2

In [12]:
```
# Train a model with 1 hidden layer and 50 neurons with 0.2 momentum

Layers = [2, 50, 3]
model = Net(Layers)
learning_rate = 0.10
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0.2)
train_loader = DataLoader(dataset=data_set, batch_size=20)
criterion = nn.CrossEntropyLoss()
Results["momentum 0.2"] = train(data_set, model, criterion, train_loader, optimizer, e
plot_decision_regions_3class(model, data_set)
```

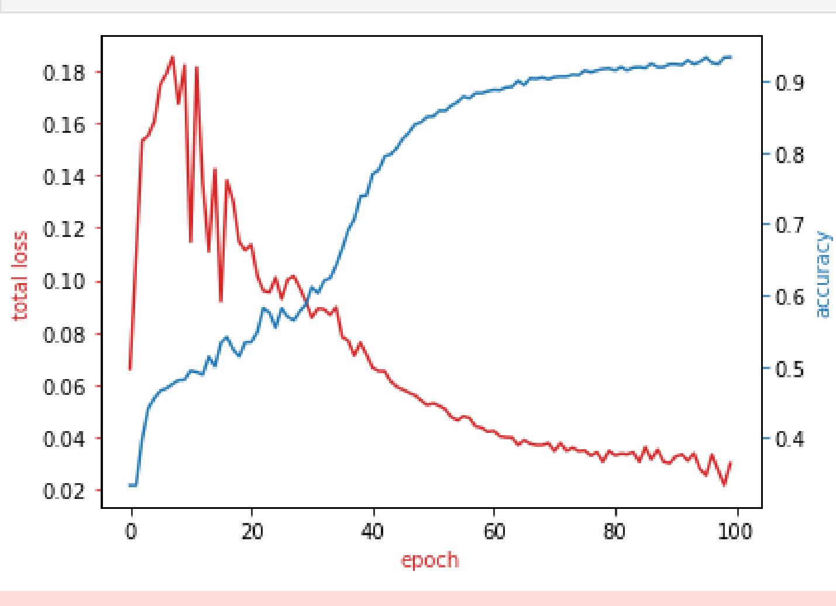


```
<ipython-input-3-f58ec7e51729>:15: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corne
rs of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud',
or set rcParams['pcolor.shading'].  This will become an error two minor releases late
r.
  plt.pcolormesh(xx, yy, yhat, cmap=cmap_light)
```

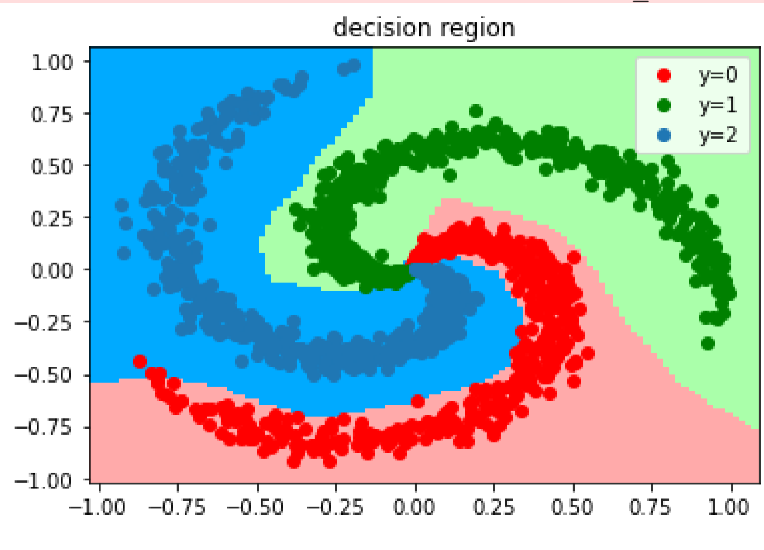


Red para clasificar 3 clases con 1 capa oculta con 50 neuronas y un momento de 0.4

In [13]:
```
# Train a model with 1 hidden layer and 50 neurons with 0.4 momentum

Layers = [2, 50, 3]
model = Net(Layers)
learning_rate = 0.10
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0.4)
train_loader = DataLoader(dataset=data_set, batch_size=20)
criterion = nn.CrossEntropyLoss()
Results["momentum 0.4"] = train(data_set, model, criterion, train_loader, optimizer, e
plot_decision_regions_3class(model, data_set)
```

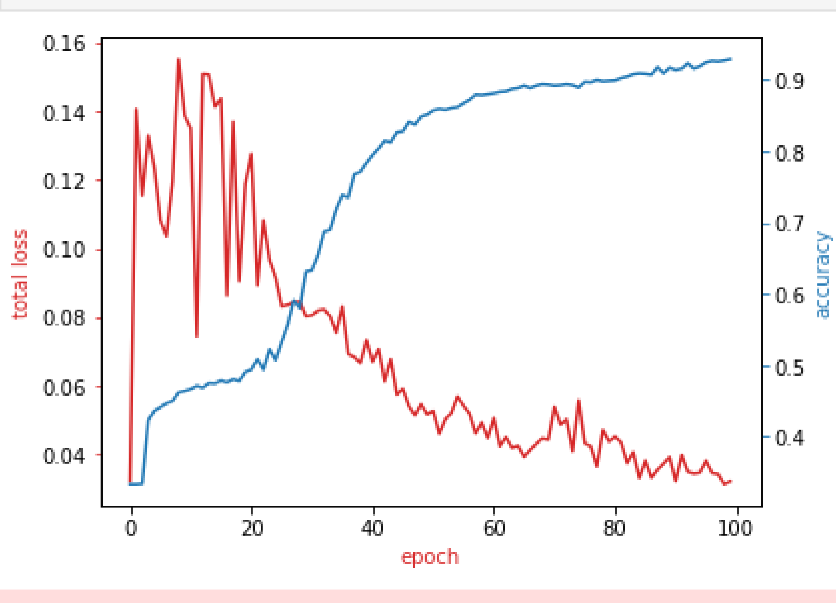


```
<ipython-input-3-f58ec7e51729>:15: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corne
rs of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud',
or set rcParams['pcolor.shading'].  This will become an error two minor releases late
r.
  plt.pcolormesh(xx, yy, yhat, cmap=cmap_light)
```

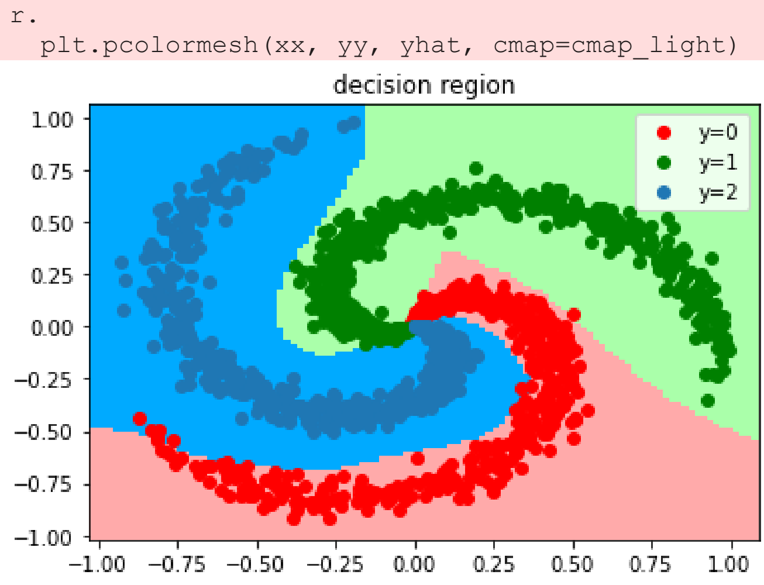


Red para clasificar 3 clases con 1 capa oculta con 50 neuronas y un momento de 0.5

In [14]:
```
# Train a model with 1 hidden layer and 50 neurons with 0.5 momentum

Layers = [2, 50, 3]
model = Net(Layers)
learning_rate = 0.10
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0.5)
train_loader = DataLoader(dataset=data_set, batch_size=20)
criterion = nn.CrossEntropyLoss()
Results["momentum 0.5"] = train(data_set, model, criterion, train_loader, optimizer, e
plot_decision_regions_3class(model,data_set)
```



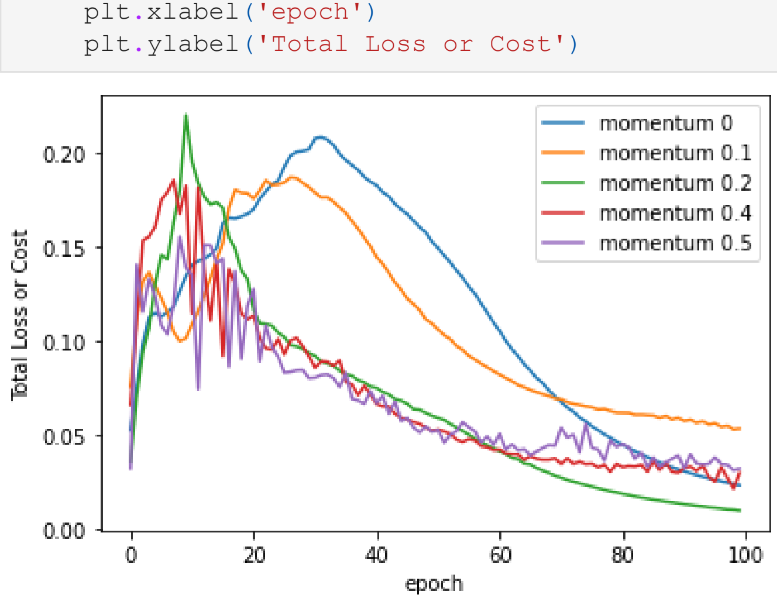
```
<ipython-input-3-f58ec7e51729>:15: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corne
rs of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud',
or set rcParams['pcolor.shading'].  This will become an error two minor releases late
r.
  plt.pcolormesh(xx, yy, yhat, cmap=cmap_light)
```



Comparamos los resultados

El gráfico de abajo compara los diferentes términos de momento. Vemos que en general el costo decrece proporcionalmente al término de momento, pero valores más grandes del momento llevan a mayores oscilaciones. En este caso, el valor de 0.2. parece ser el mejor (alcanza el menor costo)

In [15]:
```
# Plot the Loss result for each term

for key, value in Results.items():
    plt.plot(value['Loss'],label=key)
    plt.legend()
    plt.xlabel('epoch')
    plt.ylabel('Total Loss or Cost')
```



La precisión parece ser proporcional al momento:

In [16]:
```
# Plot the Accuracy result for each term

for key, value in Results.items():
    plt.plot(value['Accuracy'],label=key)
    plt.legend()
    plt.xlabel('epoch')
    plt.ylabel('Accuracy')
```