```
#required libraries

import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.colors
import time

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error, log_loss
from tqdm import tqdm_notebook

from IPython.display import HTML
import warnings
from sklearn.preprocessing import OneHotEncoder
from sklearn.datasets import make_blobs

import torch
warnings.filterwarnings('ignore')
```

```
torch.manual_seed(0)
```

```
<torch._C.Generator at 0x7f1bce072090>
```

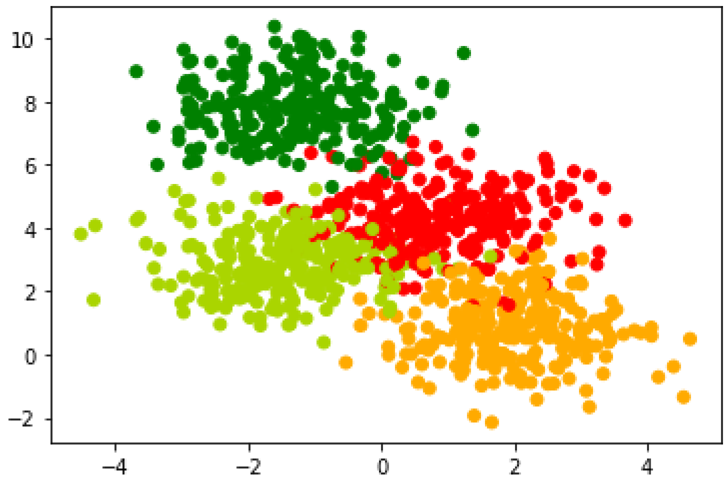
```
my_cmap  = matplotlib.colors.LinearSegmentedColormap.from_list("" , ["red" , "yellow" , "green"])
```

Generate dataset

```
data  ,labels = make_blobs(n_samples = 1000 , centers  =4  , n_features=  2  ,random_state=  0 )
print(data.shape , labels.shape)
```

```
(1000, 2) (1000,)
```

```
plt.scatter(data[:  , 0 ] ,  data[:  ,1]  , c = labels , cmap =  my_cmap )
plt.show()
```




```
X_train  ,X_val  , Y_train , Y_val  = train_test_split(data  , labels  , stratify  = labels , random_state =  0)
```

```
print(X_train.shape , Y_train.shape ,  labels.shape )
```

```
(750, 2) (750,) (1000,)
```

Using torch tensors and autograd

```
X_train , Y_train   ,X_val  , Y_val   = map(torch.tensor  , (X_train , Y_train , X_val , Y_val ))
print(X_train.shape , Y_train.shape)
```

```
torch.Size([750, 2]) torch.Size([750])
```

```
def model(x):
    a1 = torch.matmul(x, weights1) + bias1 # (N, 2) x (2, 2) -> (N, 2)
    h1 = a1.sigmoid() # (N, 2)
    a2 = torch.matmul(h1, weights2) + bias2 # (N, 2) x (2, 4) -> (N, 4)
    h2 = a2.exp()/a2.exp().sum(-1).unsqueeze(-1) # (N, 4)
    return h2
```

```
y_hat  = torch.tensor([[0.1 , 0.2, 0.3 , 0.4] , [0.8 , 0.1 , 0.05  , 0.05]])
y  = torch.tensor([2, 0])
(-y_hat[range(y_hat.shape[0]) , y ].log()).mean().item()
(torch.argmax(y_hat ,dim  =1 ) == y).float().mean().item()
```

```
0.5
```

```
def loss_fn(y_hat, y):
    return -(y_hat[range(y.shape[0]), y].log()).mean()
```

```
def accuracy(y_hat ,  y ) :
  pred = torch.argmax(y_hat , dim  =1)
  return (pred  == y ).float().mean()
```

```
plt.style.use("seaborn")
```

```
torch.manual_seed(0)

#initialize the weights and biases using He Initialization
weights1 = torch.randn(2, 2) / math.sqrt(2)
weights1.requires_grad_()
bias1 = torch.zeros(2, requires_grad=True)

weights2 = torch.randn(2, 4) / math.sqrt(2)
weights2.requires_grad_()
bias2 = torch.zeros(4, requires_grad=True)

#set the parameters for training the model
learning_rate = 0.2
epochs = 10000

X_train = X_train.float()
Y_train = Y_train.long()
X_val = X_val.float()
Y_val = Y_val.long()

loss_arr = []
acc_arr = []
val_acc_arr = []

#training the network
for epoch in range(epochs):
    y_hat = model(X_train)  #compute the predicted distribution
    loss = loss_fn(y_hat, Y_train) #compute the loss of the network
    loss.backward() #backpropagate the gradients
    loss_arr.append(loss.item())
    acc_arr.append(accuracy(y_hat, Y_train))

    with torch.no_grad(): #update the weights and biases
        val_acc_arr.append(accuracy(model(X_val),Y_val))

        weights1 -= weights1.grad * learning_rate
        bias1 -= bias1.grad * learning_rate
        weights2 -= weights2.grad * learning_rate
```

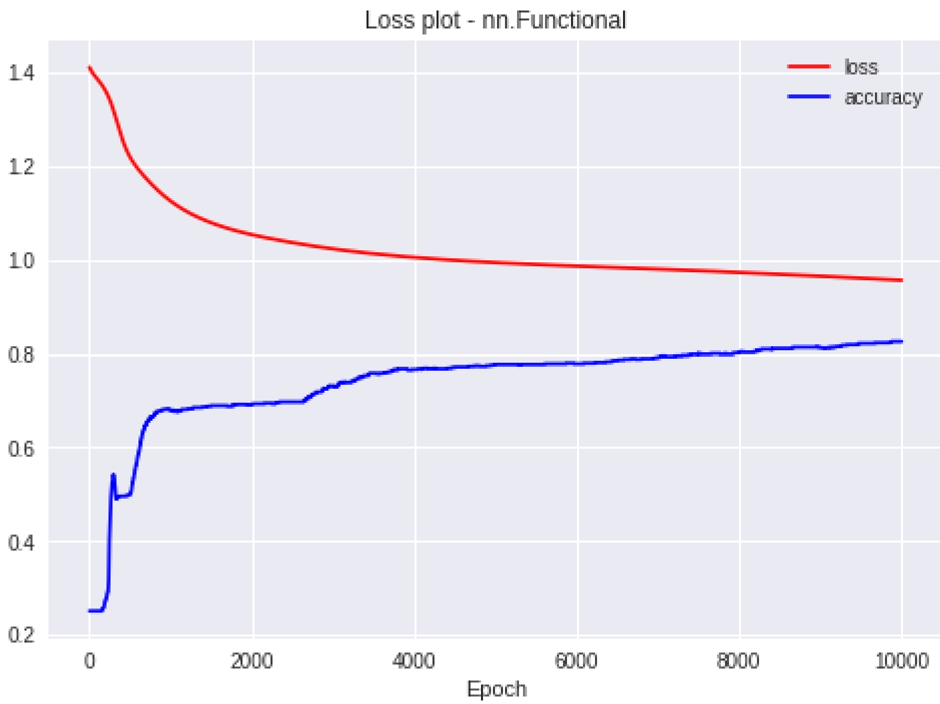
```
        bias2 -= bias2.grad * learning_rate
        weights1.grad.zero_()
        bias1.grad.zero_()
        weights2.grad.zero_()
        bias2.grad.zero_()

plt.plot(loss_arr, 'r-', label='loss')
plt.plot(acc_arr, 'b-', label='train accuracy')
plt.plot(val_acc_arr, 'g-', label='val accuracy')
plt.title("Loss plot - Using tensors and autograd")
plt.xlabel("Epoch")
plt.legend(loc='best')
plt.show()
print('Loss before training', loss_arr[0])
print('Loss after training', loss_arr[-1])
```

Using nn Functional

```
import  torch.nn.functional as F
```

```
torch.manual_seed(0)
weights1 = torch.randn(2,2) / math.sqrt(2)
weights1.requires_grad_()
bias1  = torch.zeros(2, requires_grad = True)

weights2 = torch.randn(2,4) / math.sqrt(2)
weights2.requires_grad_()
bias2 = torch.zeros(4 , requires_grad = True)


learning_rate = 0.2
epochs = 10000

loss_arr = []
acc_arr =  []


for epoch in range(epochs) :
  y_hat =  model(X_train)
  loss = F.cross_entropy(y_hat , Y_train)
  loss.backward()
  loss_arr.append(loss.item())
  acc_arr.append(accuracy(y_hat , Y_train))

  with torch.no_grad():
    weights1-= learning_rate*weights1.grad
    bias1 -= bias1.grad*learning_rate
    weights2-= learning_rate*weights2.grad
    bias2 -= bias2.grad*learning_rate
```

```
        weights1.grad.zero_()
        bias1.grad.zero_()
        weights2.grad.zero_()
        bias2.grad.zero_()

    plt.plot(loss_arr ,  "r-" , label = "loss")
    plt.plot(acc_arr , "b-" ,  label =  "accuracy")
    plt.legend(loc = "best")
    plt.title("Loss plot - nn.Functional")
    plt.xlabel("Epoch")
    plt.show()

    plt.show()
    print('Loss before training', loss_arr[0])
    print('Loss after training', loss_arr[-1])
```




```
Loss before training 1.4111980199813843
Loss after training 0.9561843276023865
```

Using NN Parameters

```
import torch.nn as nn
```

```
class FirstNetwork(nn.Module):
  def  __init__(self) :
    super().__init__()
    torch.manual_seed(0)
    self.weights1 =  nn.Parameter(torch.randn(2,2) / math.sqrt(2))
    self.bias1 = nn.Parameter(torch.zeros(2))
    self.weights2 =  nn.Parameter(torch.randn(2,4) / math.sqrt(2))
    self.bias2 = nn.Parameter(torch.zeros(4))


  def forward(self ,X):
   a1 = torch.matmul(X , self.weights1 )  + self.bias1
   h1 =a1.sigmoid()
   a2 = torch.matmul(h1 , self.weights2) + self.bias2
   h2 = a2.exp()/a2.exp().sum(-1).unsqueeze(-1)
   return h2


def fit(epochs = 10000 , learning_rate = 0.2 , title = "") :
  loss_arr = []
  acc_arr = []
  for epoch in range(epochs):
    y_hat = model(X_train)
    loss = F.cross_entropy(y_hat , Y_train)
    loss_arr.append(loss.item())
    acc_arr.append(accuracy(y_hat , Y_train))
    loss.backward() #backpropogation

    with torch.no_grad():
      for param in model.parameters():
        param -= learning_rate*param.grad
```

```
            param -= learning_rate*param.grad
          model.zero_grad()

      plt.plot(loss_arr , "r-" ,label =  "loss")
      plt.plot(acc_arr , "b-" ,label = "train_accuracy")
      plt.legend()
      plt.title(title)
      plt.xlabel("Epochs")
      plt.show()
      print("Loss before training" , loss_arr[0])
      print("Loss after training" , loss_arr[-1])


model = FirstNetwork()

fit(10000 , 0.2 , "Loss plot - nn.Parameter & nn.Module")
```

Loss plot - nn Parameter & nn Module

Using Optim and NN linear

```
class FirstNetwork_v1(nn.Module):
  def __init__(self) :
    super().__init__()
    torch.manual_seed(0)
    self.lin1 = nn.Linear(2,2)
    self.lin2 = nn.Linear(2,4)


  def forward(self ,X):
    a1 = self.lin1(X)
    h1 = a1.sigmoid()
    a2 = self.lin2(h1)
    h2 = a2.exp() / a2.exp().sum(-1).unsqueeze(-1)
    return h2
```

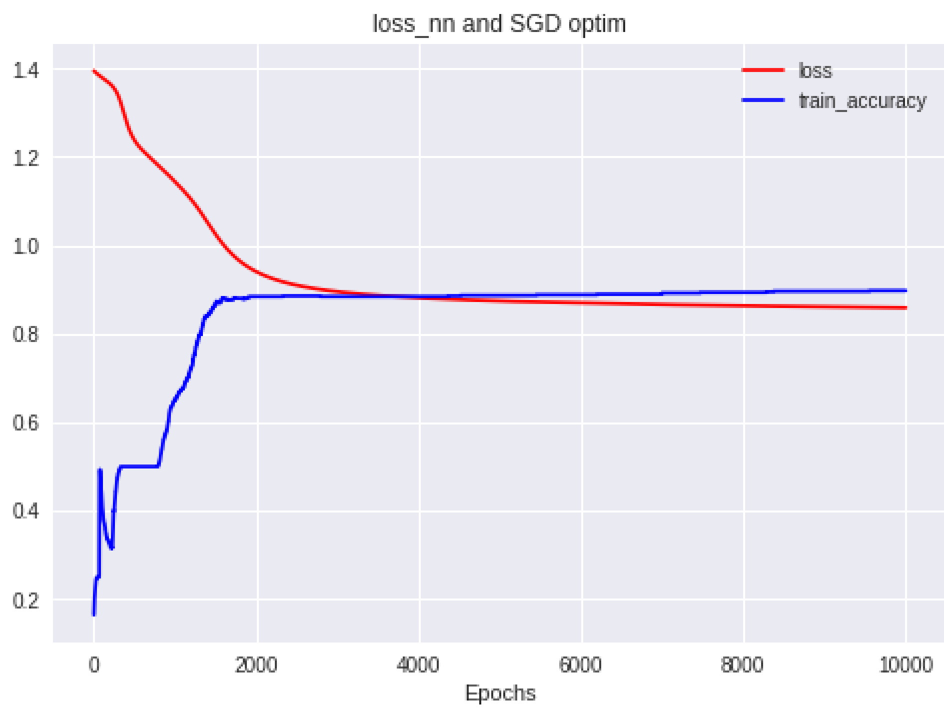
```
model = FirstNetwork_v1()

fit(10000 , 0.2 , "Loss plot - nn.Linear")
```


```
from torch import optim
```


```
def fit_v1(epochs = 10000 , learning_rate  = 0.2 ,  title = "") :
  loss_arr = []
  acc_arr = []

  opt = optim.SGD(model.parameters() , lr =  learning_rate)

  for epoch in range(epochs):
    y_hat = model(X_train)
    loss = F.cross_entropy(y_hat , Y_train)
    loss_arr.append(loss.item())
    acc_arr.append(accuracy(y_hat  ,Y_train))


    loss.backward()
    opt.step()
    opt.zero_grad()

  plt.plot(loss_arr , "r-" , label =  "loss" )
  plt.plot(acc_arr ,  "b-" , label = "train_accuracy")
  plt.legend()
  plt.title(title)
  plt.xlabel("Epochs")
  plt.show()
  print("Loss before training " , loss_arr[0])
```

```
    print('Loss after training' , loss_arr[-1])
```

```
%%time
model = FirstNetwork_v1()


fit_v1( 10000 , 0.2 ,  "loss_nn and SGD optim")
```




```
Loss before training  1.395160436630249
Loss after training 0.8586323857307434
CPU times: user 9.31 s, sys: 48.9 ms, total: 9.36 s
Wall time: 9.36 s
```

```
%%time
```

```
%%time
model = FirstNetwork_v1()


fit_v1( 2800 , 0.2 ,  "loss_nn and adam  optim")
```


```
Loss before training  1.395160436630249
Loss after training 0.8998849987983704
CPU times: user 2.75 s, sys: 18 ms, total: 2.77 s
Wall time: 2.77 s
```

Using NN sequential

```
class FirstNetwork_v2(nn.Module):
  def __init__(self):
    super().__init__()
    torch.manual_seed(0)
    self.net = nn.Sequential(
```

```
        nn.Linear(2,2) ,
        nn.Sigmoid() ,
        nn.Linear(2,4) ,
        nn.Softmax() )
  def forward(self , X) :
    return self.net(X)
```

```
model = FirstNetwork_v2()

def fit_v2(x, y, model, opt, loss_fn, epochs = 10000):
    """Generic function for training a model """
    for epoch in range(epochs):
        loss = loss_fn(model(x), y)

        loss.backward()
        opt.step()
        opt.zero_grad()

    return loss.item()
```

```
loss_fn  =F.cross_entropy
opt  = optim.SGD(model.parameters() , lr = 0.2)

fit_v2(X_train , Y_train  ,  model , opt , loss_fn)
```

```
0.8586323857307434
```

Running on GPUs

```
device = torch.device("cuda")


X_train=X_train.to(device)
Y_train=Y_train.to(device)

model = FirstNetwork_v2()
model.to(device) #moving the network to GPU

#calculate time
tic = time.time()
print('Final loss', fit_v2(X_train, Y_train, model, opt, loss_fn))
toc = time.time()
print('Time taken', toc - tic)
```

```
Final loss 1.395159363746643
Time taken 7.821534633636475
```

```
class FirstNetwork_v3(nn.Module):

    def __init__(self):
        super().__init__()
        torch.manual_seed(0)
        self.net = nn.Sequential(
            nn.Linear(2, 1024*4),
            nn.Sigmoid(),
            nn.Linear(1024*4, 4),
            nn.Softmax())

    def forward(self, X):
        return self.net(X)
```

```
device = torch.device("cpu")

X_train=X_train.to(device)
Y_train=Y_train.to(device)

#training on gpu
fn = FirstNetwork_v3()
fn.to(device)

tic = time.time()
print('Final loss', fit_v2(X_train, Y_train, fn, opt, loss_fn))
toc = time.time()
print('Time taken', toc - tic)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-34-dee9772d0354> in <module>()
      9 
     10 tic = time.time()
---> 11 print('Final loss', fit_v2(X_train, Y_train, fn, opt, loss_fn))
     12 toc = time.time()
     13 print('Time taken', toc - tic)
```

2 frames

```
/usr/local/lib/python3.6/dist-packages/torch/autograd/__init__.py in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables)
    125     Variable._execution_engine.run_backward(
    126         tensors, grad_tensors, retain_graph, create_graph,
--> 127         allow_unreachable=True)  # allow_unreachable flag
    128 
    129 

KeyboardInterrupt: 
```

SEARCH STACK OVERFLOW