Artificial Neural Network (ANN) - Classification 1

Rafiq Islam

2025-03-25

Table of contents

**Incomplete**

## Binary Classification

Say we have a dataset like this

import torch
import numpy as np
import matplotlib.pyplot as plt
from mywebstyle import plot\_style
plot\_style('#f4f4f4')

np.random.seed(0)
n=100
center1 = [2, 2]
center2 = [0, -2]
stdv = 1.5
cluster1 = [
 center1[0] + np.random.randn(n)\*stdv, center1[1] + np.random.randn(n)\*stdv
]
cluster2 = [
 center2[0] + np.random.randn(n)\*stdv, center2[1] + np.random.randn(n)\*stdv
]

data\_matrix = np.hstack((cluster1, cluster2)).T
data = torch.tensor(data\_matrix).float()
labels = torch.tensor(
 np.vstack((np.zeros((n,1)),(np.ones((n,1)))))
).float()

plt.scatter(
 data[np.where(labels==0)[0],0],
 data[np.where(labels==0)[0],1],
 color='red',
 label = 'class 1'
)
plt.scatter(
 data[np.where(labels==1)[0],0],
 data[np.where(labels==1)[0],1],
 color='blue',
 label = 'class 2'
)
plt.legend()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()



and we want to make an ANN classifier model with this data. So, we consider a two layer neural network



So our model

import torch.nn as nn

ANN\_classifier = nn.Sequential(
 nn.Linear(2,1), # Input layer mapping R^2--> R
 nn.ReLU(), # Activation function in layer 1
 nn.Linear(1,1), # Output layer
 nn.Sigmoid() # Activation function in layer 2
)

Now let’s train the model

lr = 0.01 # Learning Rate
loss\_function = nn.BCELoss() # Binary Cross Entropy Loss
optimizer = torch.optim.SGD( # Stochastic Gradient Descent Optimizer
 ANN\_classifier.parameters(),
 lr=lr
)
num\_epochs = 1000 # Number of Epochs

# Define losses to store the loss from each epoch
losses = torch.zeros(num\_epochs)
for epoch in range(num\_epochs):
 # Forward Pass
 pred = ANN\_classifier(data)

 # Compute loss
 loss = loss\_function(pred, labels)
 losses[epoch] = loss

 # Backpropagation
 optimizer.zero\_grad()
 loss.backward()
 optimizer.step()

plt.plot(losses.detach())
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()



Next we compute the predictions made the model

preds = ANN\_classifier(data)
predicted\_labels = preds>0.5
mis\_classification = np.where(predicted\_labels != labels)[0]
acc = 100 - 100\*len(mis\_classification)/(2\*100)
plt.plot(
 data[mis\_classification,0], data[mis\_classification,1],
 'rx', markersize=12, markeredgewidth=3
)
plt.plot(
 data[np.where(~predicted\_labels)[0],0],
 data[np.where(~predicted\_labels)[0],1],'bs'
)
plt.plot(
 data[np.where(predicted\_labels)[0],0],
 data[np.where(predicted\_labels)[0],1],'ko'
)
plt.title(f'{acc}% correct')
plt.show()



## Multiclass Classification

We use the IRIS data for this project

import seaborn as sns
iris = sns.load\_dataset('iris')
sns.pairplot(iris, hue='species')
plt.show()



### Data Pre-process

data = torch.tensor(iris[iris.columns[0:4]].values).float()
labels = torch.zeros(len(data), dtype=torch.long)
labels[iris.species=='versicolor']=1
labels[iris.species=='virginica']=2

Model

iris\_classifier = nn.Sequential(
 nn.Linear(4,64),
 nn.ReLU(),
 nn.Linear(64,64),
 nn.ReLU(),
 nn.Linear(64,3)
)
loss\_fun = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(iris\_classifier.parameters(), lr=0.01)

Training

num\_epochs = 1000
losses = torch.zeros(num\_epochs)
running\_acc = []

for epoch in range(num\_epochs):
 # forward pass
 yhat = iris\_classifier(data)
 # compute loss
 loss = loss\_fun(yhat, labels)
 losses[epoch] = loss

 # back-prop
 optimizer.zero\_grad()
 loss.backward()
 optimizer.step()

 # accuracy
 matches = torch.argmax(yhat, axis=1)==labels
 matchesNum = matches.float()
 acc = 100\*torch.mean(matchesNum)
 running\_acc.append(acc)

# Model prediction
preds = iris\_classifier(data)
predicted\_labels = torch.argmax(preds, axis=1)
totalacc = 100\*torch.mean((predicted\_labels==labels).float())

fig, ax = plt.subplots(1,2, figsize=(9,4))
ax[0].plot(losses.detach())
ax[0].set\_ylabel('loss')
ax[0].set\_xlabel('epoch')
ax[0].set\_title('Losses')

ax[1].plot(running\_acc)
ax[1].set\_ylabel('accuracy')
ax[1].set\_xlabel('epoch')
ax[1].set\_title('Accuracy')
plt.show()

