Artificial Neural Network (ANN) - Regression

Rafiq Islam

2025-03-25

Table of contents

## Simple Linear Regression Using ANN

The simple linear regression equation is given as

$$y\_{i}=β\_{0}+β\_{1}x\_{i}+ξ\_{i}=σ\left(w\_{0}+x^{T}w\right)=σ\left(x^{T}w\right)$$



The loss function in this case MSE: Mean Squared Error

import torch
n = 50
# Creating n=50 random X values from the standard normal distribution
X = torch.randn(n,1)
# y = mX + c + noise. Here m=1, c = 0, noise = N(0,1)/2
y = X + torch.randn(n,1)/2

plt.plot(X,y, 'ro')
plt.xlabel('X')
plt.ylabel('y')
plt.show()



Now the model

import numpy as np
import torch.nn as nn

ANN\_regressor = nn.Sequential(
 nn.Linear(1,1), # Input Layer
 nn.ReLU(), # Rectified Linear Unit (ReLU) activation function
 nn.Linear(1,1) # Output Layer
)
ANN\_regressor

Sequential(
 (0): Linear(in\_features=1, out\_features=1, bias=True)
 (1): ReLU()
 (2): Linear(in\_features=1, out\_features=1, bias=True)
)

Next we want to train our model using *Stochastic Gradient Descent* optimizer

lr = 0.05 # Learning rate/stepsize
loss\_function = nn.MSELoss() # MSE loss function
optimizer = torch.optim.SGD( # SGD Optimizer
 ANN\_regressor.parameters(),
 lr=lr
)

training\_epochs = 500 # Epochs
losses = torch.zeros(training\_epochs) # Creating 1D zero vector of size 500

# Train the model

for epoch in range(training\_epochs):

 # forward pass
 pred = ANN\_regressor(X)

 # compute the loss
 loss = loss\_function(pred, y)
 losses[epoch] = loss

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

predictions = ANN\_regressor(X)
test\_loss = (predictions - y).pow(2).mean()

plt.plot(losses.detach())
plt.plot(training\_epochs, test\_loss.detach(), 'ro')
plt.title('Final Loss = %g' %test\_loss.item())
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()



Now let’s calculate the predictions

plt.plot(X,y, 'ro', label = 'Data')
plt.plot(X,predictions.detach(), 'bs', label='Predictions')
plt.legend()
plt.show()



Putting all together

def ann\_reg(X,y):
 model = nn.Sequential(
 nn.Linear(1,1),
 nn.ReLU(),
 nn.Linear(1,1)
 )
 loss\_function = nn.MSELoss()
 optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
 training\_epochs = 500

 losses = torch.zeros(training\_epochs)

 for epoch in range(training\_epochs):
 pred = model(X)

 loss = loss\_function(pred, y)
 losses[epoch] = loss

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

 return model(X), losses

def data(m):
 X = torch.randn(50,1)
 y = m\*X + torch.randn(50,1)/2

 return X, y

slopes = np.linspace(-2,2,21)

train = 30

results = np.zeros((len(slopes), train,2))

for m in range(len(slopes)):
 for t in range(train):
 X,y = data(slopes[m])
 prediction,loss = ann\_reg(X,y)
 results[m, t, 0] = loss[-1]
 results[m, t, 1] = np.corrcoef(y.T,prediction.detach().T)[0,1]

results[np.isnan(results)]=0

fig, ax = plt.subplots(1,2, figsize=(8,4))

ax[0].plot(slopes, np.mean(results[:,:,0], axis=1),'ko-')
ax[0].set\_xlabel('Slope')
ax[0].set\_title('Loss')

ax[1].plot(slopes, np.mean(results[:,:,1],axis=1),'ms-')
ax[1].set\_xlabel('Slope')
ax[1].set\_ylabel('Real vs Predicted correlation')
ax[1].set\_title('Model Performance')
plt.show()



**Share on**

**You may also like**