Amirkabir University of Technology
(Tehran Polytechnic)

Electrical Engineering Department

گزارشکار آزمایشگاه مقدمه‌ای بر هوش محاسباتی

آزمایش شماره‌های 3

MultiLayer Perceptron

نام استاد: محمدحسین امینی

نام دانشجو: محمدعرشیا ثمودی - 9723021

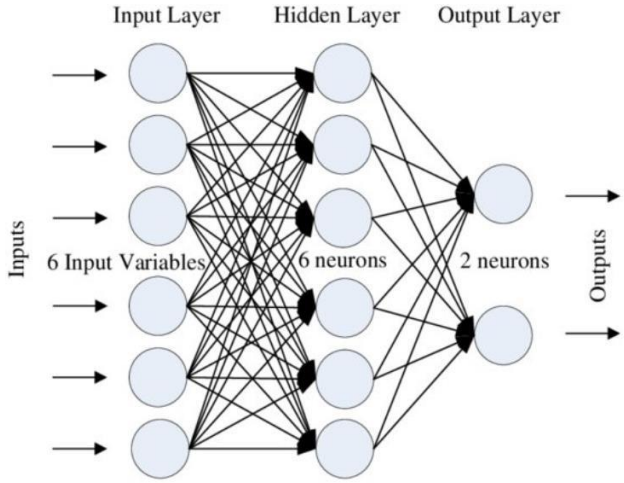
آزمایش سوم

هدف آزمایش: پیاده‌سازی مدل شبکه عصبی چندلایه

شرح آزمایش:

1. مقدمه

مدل‌های شبکهSLP دارای مشکلات زیاد و محدودیت در استفاده می‌باشند که از جمله آنها می‌توان به ناتوانی آن در مقابله با غیرخطی بودن و محدودیت در مقدار و تعداد خروجی اشاره کرد که راه حل این مشکلات شبکه‌های MLP هستند. شبکه SLP توانایی تشخیص داده‌های جداپذیر خطی را داشت.



این شبکه از قسمت های مختلفی تشکیل شده‌اند:

1. لایه‌ی ورودی (input layer)
2. لایه‌ی مخفی (hidden layer)
3. لایه‌ی خروجی (output layer)
4. وزن ها و بایاس ها
5. توابع فعال ساز

یادگیری در شبکه عصبی با تغییر وزن اتصال پس از پردازش هر قطعه از داده‌ها، براساس میزان خطا در خروجی در مقایسه با نتیجه مورد انتظار رخ می‌دهد.

روش های بسیاری برای استفاده از MLP وجود دارد که یکی از مهم ترین آنها روش Back-Propagation Training یا روش پس انتشار خطا است. در این روش، در هر دور (یعنی در هر تکرار) دو مرحله خواهیم داشت. مرحله‌ی اول حرکت رو به جلو (feed forward) است که، با ضربِ داده‌های ورودی در وزن‌ها و سپس جمع آن با انحراف انجام می‌شود.

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j} e_j^2(n)$$

با استفاده از گرادیان، تغییر در وزن به صورت زیر است:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n))\sum_{k} -\frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n)$$

سرانجام در همان مرحله‌ی اول به یک خروجی می‌رسیم که احتمالاً با خروجیِ واقعی تفاوت دارد. اینجاست که توسط تابع ضرر مشخص می‌کنیم که مرحله‌ی feed forward چه مقدار خطایی داشته است. حال که فهمیدیم الگوریتم با توجه به وزن‌ها و انحراف‌ها چه مقدار خطایی دارد، به مرحله‌ی دوم در یک تکرار می‌رویم. در این مرحله می‌توانیم به عقب بازگشته و وزن‌ها و انحراف‌ها را به‌هنگام‌سازی کنیم.

یعنی وزن‌ها و انحراف‌ها را به شکلی تغییر دهیم تا در تکرارِ بعدی نتیجه‌ای نزدیک‌تر به خروجیِ واقعی و با خطای کم‌تر را تولید کنند. این تکرار ها آنقدر انجام می‌شود تا خروجیِ شبکه برای تمامیِ داده‌های آموزشی، به نزدیک‌ترین

مقدارِ واقعیِ خود (یعنی مقداری که توسط داده‌های آموزشی در اختیار داریم) برسد. به این ترتیب الگوریتم یاد می‌گیرد.

در قسمت ابتدائی، بردار وزن w1 و w2 و همچنین بایاس‌های b1 و b2 را تعریف می‌کنیم.

```
class MLP:

  def __init__(self,x,y,hidden_neurons = 5):
    self.x = x
    self.y = y
    self.hidden_neurons = hidden_neurons
    self.w1 = np.random.rand(x.shape[0],hidden_neurons)
    self.b1 = np.random.rand(1,hidden_neurons)
    self.w2 = np.random.rand(hidden_neurons,y.shape[0])
    self.b2 = np.random.rand(1,y.shape[0])
```

سپس در حلقه ای به ازای تعداد epochs های ورودی، خروجی لایه‌ی اول را با ضرب وزن ها در ورودی و عبور از تابع فعال ساز بدست می‌آوریم. حال خروجی لایه‌ی دوم را نیز بدست آورده و با محاسبه‌ی خروجی نهایی مسیر feedforward را کامل میکنیم.

```
  def fit(self,iteration = 10000):

    for iter in range(iteration):
      self.costfunc = 0
      z = np.dot(self.x,self.w1) + self.b1
      hiddenlayeroutput = sigmoid(z)
      z = np.dot(hiddenlayeroutput,self.w2) + self.b2
      output = sigmoid(z)

      # calculating cost function
      self.costfunc = 0.5* np.sum(np.power((self.y-output),2))
      outputerror = 0.5 * (self.y - output) * (1- np.power(output,2))
      hiddenerror = 0.5 * (1 -np.power(hiddenlayeroutput,2))* (outputerror * self.w2.T)

      # update weights
      ar = 0.1
      self.w2 += ar*np.dot(hiddenout.T ,outputerror)
      self.w1 += ar*np.dot(self.x.T , hiddenerror)

      # update biases
      self.b2 += np.sum(outputerror) * ar
      self.b1 += np.sum(hiddenerror) * ar

      if costfunc < 0.01:
        break;
```

حال به کامل کردن مسیر فیدبک میپردازیم، مطابق بلوک دیاگرام بالا وزن های دو لایه را آپدیت میکنیم در حلقه تابع هزینه هم محاسبه میشود تا اگر به دقت خوبی برسد از حلقه خارج شود.

```
def predict(self,xpredict):
  hiddenoutput_predict = self.sigmoid(np.dot(xpred.self.w1)+self.b1)
  pred = selt.sigmoid(np.dot(hiddenoutput_predict,self.w2)+ self.b2)
  return pred.T
```