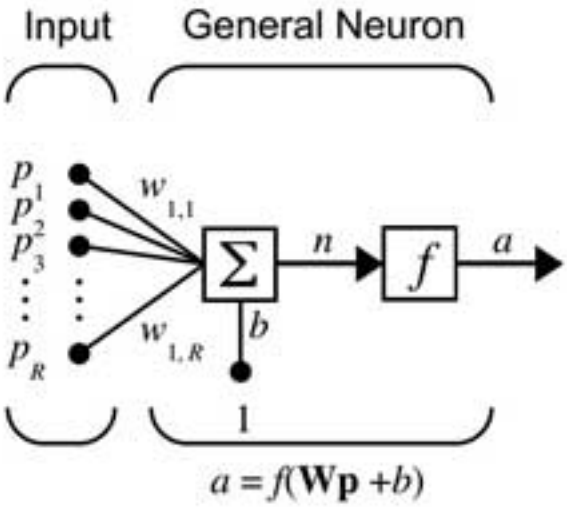
الگوریتم Backpropagation

شبکه پس انتشار (Back Propagation Netwok)، نوعی شبکه عصبی چندلایه با تابع انتقال غیرخطی و قاعده یادگیری Widrow-Hoff می‌باشد. از بردار ورودی و هدف در راستای آموزش این نوع شبکه استفاده می شود.

مرسوم‌ترین معماری برای شبکه پس‌انتشار، شبکه Feed Forward می‌باشد. یک نورون ساده با R ورودی در شکل زیر نشان داده شده:



شبکه‌های Feed Forward، اغلب دارای یک یا چندلایه مخفی از نورون‌ها بوده که اصولا برای فعال سازی آن ها از تابع Sigmoid استفاده می کنند و از یک‌لایه پایانی خطی استفاده می‌کند. وجود چند لایه از نورون‌ها با یک تابع انتقال غیرخطی به شبکه اجازه می‌دهد که توانایی یادگیری خطی و غیرخطی بین ورودی‌ها و خروجی‌ها داشته باشد.

الگوریتم آپدیت وزن ها در این الگوریتم به صورت زیر می باشد:
در هر لایه اختلاف خروجی و تارگت را در مشتق تابع فعال سازی ضرب می کنیم و این مقدار را به وزن هر لایه اضافه می کنیم.
در این تمرین شرط توقف الگوریتم زمانی است که telorance = 0.2 شود به این معنی که اختلاف هر نرون خروجی با تارگت کمتر از ۰.۲ باشد.
الگوریتم اجرا شده در این تمرین یک الگوریتم Autoassociative است که با ورودی و خروجی یکسان وزن های لایه میانی را تعین می کند.
ورودی های ما ۱۰ کاراکتر اول الفبا هستند که به صورت ماتریس ۹*۷ از درایه های ۰ و ۱ نشان داده شده اند و در هر بار اجرای الگوریتم یک کاراکتر به عنوان ورودی و همچنین خروجی در نظر گرفته می شود.

تابع الگوریتم back propagation :

```
def algorithm (x, weight1, weight2, learn_rate):
    value = []
    layer1 = np.c_[np.ones(1), [x]]
    layer2 = np.c_[np.ones(1), vecmoid(layer1.dot(weight1))]
    layer3 = sigmoid(layer2.dot(weight2))
    delta3 = x - layer3
    delta2 = np.multiply(delta3.dot(weight2.T), np.multiply(layer2, (1-layer2)))
    delta2 = delta2[:,1:]
    weight2 += learn_rate * (np.dot(layer2.T, delta3))
    weight1 += learn_rate * (np.dot(layer1.T, delta2))
```

در این تابع layer1 لایه اول نرون های ورودی و نرون بایاس را ذخیره می کند.

layer2 از ضرب ماتریسی ورودی در ماتریس وزن و اعمال تابع sigmoid ایجاد می شود.

و layer3 هم از ضرب لایه دوم در ماتریس وزن دوم و اعمال تابع سیگموید به دست می آید که هدف ما همگرا کردن این لایه به ورودی است.
سپس بعد از محاسبه ی لایه ها وزن ها با همان الگوریتم گفته شده آپدیت می شوند و تا زمانی که به شرط توقف الگوریتم نرسیده باشیم این الگوریتم ادامه خواهد داشت.

تابع train():

```
def train(X):

    graph = np.zeros((1000,2), dtype=float)
    plot = np.zeros((31,2), dtype=float)
    for i in range (8,31):
        e = 0
        weight1 =np.random.uniform(low=-0.5, high=0.5, size=(64,i))
        weight2 =np.random.uniform(low=-0.5, high=0.5, size=(i+1,63))
        while True :
            for l in range(10):
                x = X[l]
                algorithm (x, weight1, weight2, learn_rate)
            tol = 0
            tolerance = []
            value = []
            for j in range(10):
                layer1 = np.c_[np.ones(1), [X[j]]]
                layer2 = np.c_[np.ones(1), vecmoid(layer1.dot(weight1))]
                layer3 = sigmoid(layer2.dot(weight2))
                value.append(layer3)
                tolerance = np.subtract (value[j],X[j])
                for k in range (63):
                    if  -0.2 <= tolerance[0][k] <= 0.2:
                        tol += 1
            if tol == 630:
                break
            dis = np.subtract(value[0], X[0])
            error = np.inner(dis , dis)
```

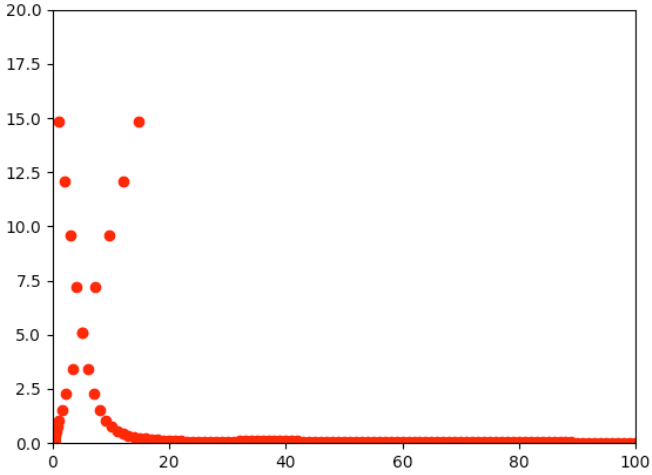
در این تابع ابتدا یک لوپ به ازای تعداد نرون های متفاوت لایه ی نهان در بازه ی (۸,۳۱) ایجاد می کنیم و به ازای هر تعداد مشخص شده مقدار weight1 و weight2 را متناظر با آن به صورت رندوم در بازه ی (-0.5,0.5) ایجاد می کنیم. (اندازه ی بازه ی وزن تاثیر بسزایی در سریع کردن روند همگرایی دارد و در این الگوریتم هر چقدر کران های این بازه به صفر نزدیک تر باشند تابع سیگموید با سرعت بیشتری تغیر می کند)

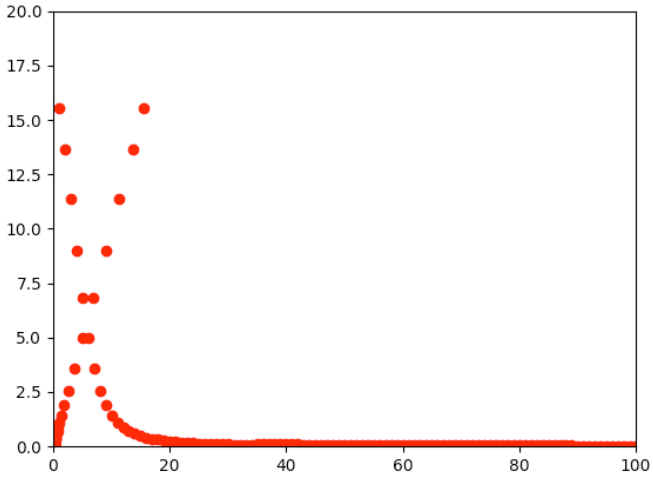
سپس تا وقتی که شرط توقف برقرار نشده این مراحل اجرا می شود:

۱-به ازای تمام کاراکتر ها ماتریس وزن با تابع algorithm آپدیت می شود.

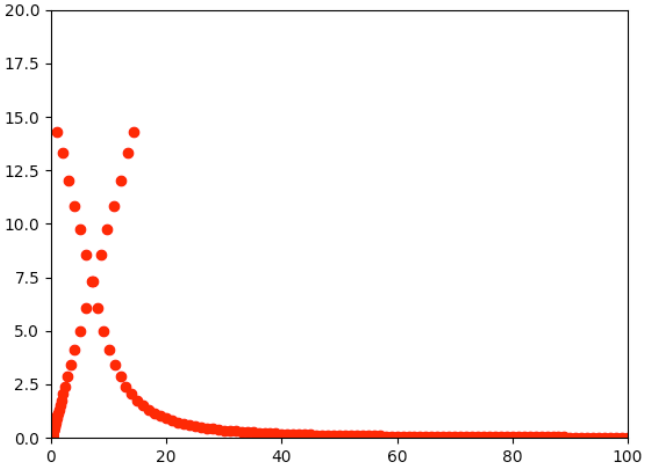
۲-به ازای تمام کاراکتر ها اختلاف خروجی مورد نظر و خروجی این شبکه محاسبه می شود و در صورتی که به ازای تمام نرون های هر کاراکتر این اختلاف کمتر از ۰.۲ باشد الگوریتم متوقف می شود در غیر این صورت به مقدار epoch یک واحد اضافه می شود و square error کاراکتر اول را به عنوان نمونه محاسبه می کنیم.

مقدار error به ازای ۱۰ , ۲۰ , ۳۰ نرون لایه های پنهان در زیر نشان داده شده است:





همانطور
که پیداست با افزایش نرون ها تعداد epoch های برای همگرا شدن میزان خطا به صفر کمتر می شود.



و نمودار زیر نشان دهنده تعداد epoch های لازم برای برقرای شرط توقف (tolerance<0.2) به ازای تعداد نرون های متفاوت در بازه ی ۸ تا ۳۰ است.