

## ”به نام یزدان پاک“

گزارش کار فصل شبکه عصبی Fruit classification در کل این آزمایش برای پیدا کردن نوع میوه ها با استفاده از شبکه عصبی است.

قدم اول:

برای خواندن دیتا ست که کد آن در گزارش کار در اختیارمان قرار گرفت.

قدم دوم:

تنهای Feed Forward است که با استفاده از لیست رندوم که تولید میکند حدس میزنند میوه چیست و در نهایت accuracy آن را پیدا میکند اما از آن جایی که اعداد کاملاً رندوم اند و train نداشته اند حدود 25% بدست میاید.

```
73     # random the weight and bias of layers
74     W1 = np.random.normal(size=(150,102))
75     W2 = np.random.normal(size=(60, 150))
76     W3 = np.random.normal(size=(4, 60))
77     b1 = np.ones((150, 1))
78     b2 = np.ones((60, 1))
79     b3 = np.ones((4, 1))
80
81     # find the sigmoid of each node with formula 1/(1+e^(-W+sig last node + bias))
82     for i in range_(number_of_train):
83         sig_y = 1/(1 + pow(np.e, -(W1 @ train_set[i][0] + b1)))
84         sig_z = 1 / (1 + pow(np.e, -(W2 @ sig_y + b2)))
85         sig_c = 1 / (1 + pow(np.e, -(W3 @ sig_z + b3)))
86
87     # find the accuracy of nodes by counting how many are fruits were guesses true
88     counter = 0
89     for i in range_(number_of_train):
90         label = train_set[i][1]
91         if sig_c.tolist().index(max(sig_c)) == label.tolist().index(max(label)):
92             counter+=1
93     print('Accuracy: ', counter/number_of_train)
94
```

### قدم سوم:

است که به تعداد epoch ها و با batch size epoch روى شبکه عصبی back propagation میزند تا بتواند نوع میوه را به طور دقیق تری حس بزنده کاملاً رندوم در نهایت مقدار propagation عددی حدوداً بین 30%- 70% (پایین تر از 70% است زیرا با این مقدار train دادن حس با دقت بالایی نمیتوان داشت)

```
81 # with some epochs and batches find the cost and accuracy we can say we do
82 # back propagation epoch's time so the result can be more accurate with for loop
83 for epoch in range(number_of_epochs): #5
84     random.shuffle(train_set)
85     train_set1 = train_set[:number_of_train]
86     for batch in range_(batch_size): #10
87         # random the weight and bias of layers
88         W1_grad = np.zeros((150, 102))
89         W2_grad = np.zeros((60, 150))
90         W3_grad = np.zeros((4, 60))
91         b1_grad = np.zeros((150, 1))
92         b2_grad = np.zeros((60, 1))
93         b3_grad = np.zeros((4, 1))
94
95         # gradient descent to upgrade the wight and bias so the accuracy can be higher
96         for k in range_(batch_num): #20
97             image = train_set1[batch * 20 + k][0]
98             label_grad = train_set1[batch * 20 + k][1]
99             sig_y_grad = 1 / (1 + pow(np.e, -(W1 @ image + b1)))
100            sig_z_grad = 1 / (1 + pow(np.e, -(W2 @ sig_y_grad + b2)))
101            sig_c_grad = 1 / (1 + pow(np.e, -(W3 @ sig_z_grad + b3)))
102
103            # weight
104            for j in range(W3_grad.shape[0]):
105                for k in range(W3_grad.shape[1]):
106                    W3_grad[j, k] += 2 * (sig_c_grad[j, 0] - label_grad[j, 0]) * sig_c_grad[j, 0] * (1 - sig_c_grad[j, 0]) * sig_z_grad[k, 0]
107
108            # bias
109            for j in range(b3_grad.shape[0]):
110                b3_grad[j, 0] += 2 * (sig_c_grad[j, 0] - label_grad[j, 0]) * sig_c_grad[j, 0] * (1 - sig_c_grad[j, 0])
111
112            #3rd layer
113            delta2 = np.zeros((60, 1))
114            for k in range(60):
115                for j in range(4):
116                    delta2[k, 0] += 2 * (sig_c_grad[j, 0] - label_grad[j, 0]) * sig_c_grad[j, 0] * (1 - sig_c_grad[j, 0]) * W3[j, k]
117
118            # weight
119            for k in range(W2_grad.shape[0]):
120                for m in range(W2_grad.shape[1]):
121                    W2_grad[k, m] += delta2[k, 0] * sig_z_grad[k, 0] * (1 - sig_z_grad[k, 0]) * sig_y_grad[m, 0]
122
123            # bias
124            for k in range(b2_grad.shape[0]):
125                b2_grad[k, 0] += delta2[k, 0] * sig_z_grad[k, 0] * (1 - sig_z_grad[k, 0])
126
127            # 2nd layer
128            delta1 = np.zeros((150, 1))
129            for m in range(150):
130                for k in range(60):
131                    delta1[m, 0] += delta2[k, 0] * sig_z_grad[k, 0] * (1 - sig_z_grad[k, 0]) * W2[k, m]
132
133            # weight
134            for m in range(W1_grad.shape[0]):
135                for v in range(W1_grad.shape[1]):
136                    W1_grad[m, v] += delta1[m, 0] * sig_y_grad[m, 0] * (1 - sig_y_grad[m, 0]) * image[v, 0]
137
138            # bias
139            for m in range(b1_grad.shape[0]):
140                b1_grad[m, 0] += delta1[m, 0] * sig_y_grad[m, 0] * (1 - sig_y_grad[m, 0])
141
142            # upgrade the weights and layers
143            W3 = W3 - (learning_rate * (W3_grad / batch_size))
144            W2 = W2 - (learning_rate * (W2_grad / batch_size))
145            W1 = W1 - (learning_rate * (W1_grad / batch_size))
```

Search Everywhere Double ⌂  
Searches for:  
- Classes  
- Files  
- Tool Windows  
- Actions  
- Settings

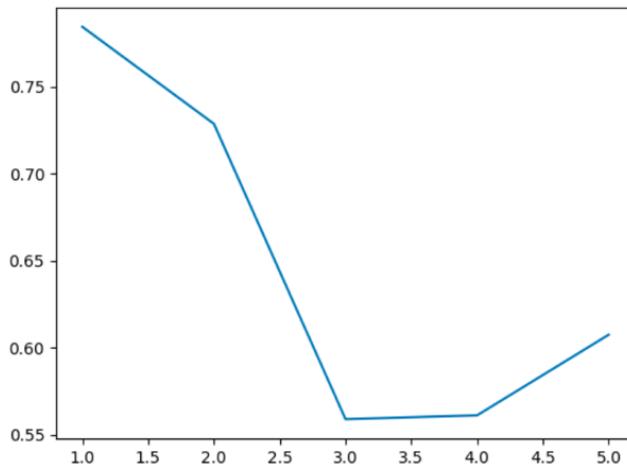
Looks like you're using NumPy  
Would you like to turn scientific mode on?

```

146     b3 = b3 - (learning_rate * (b3_grad / batch_size))
147     b2 = b2 - (learning_rate * (b2_grad / batch_size))
148     b1 = b1 - (learning_rate * (b1_grad / batch_size))
149
150     cost = 0
151     for train_data in train_set[:number_of_train]:      #200
152         x_grad = train_data[0]
153         sig_y_grad = 1 / (1 + pow(np.e, -(W1 @ x_grad + b1)))
154         sig_z_grad = 1 / (1 + pow(np.e, -(W2 @ sig_y_grad + b2)))
155         sig_c_grad = 1 / (1 + pow(np.e, -(W3 @ sig_z_grad + b3)))
156         for j in range(4):
157             cost += np.power((sig_c_grad[j, 0] - train_data[1][j, 0]), 2)
158         label_grad = train_data[1]
159     cost /= number_of_train
160     total_costs.append(cost)
161
162     # show the plot of epoch size and total cost of each
163     epoch_size = [x*1 for x in range(number_of_epochs)]
164     plt.plot(epoch_size, total_costs)
165     plt.show()
166
167     # find the accuracy of train
168     counter = 0
169     for train_data in train_set[:number_of_train]:
170         x_grad = train_data[0]
171         sig_y_grad = 1 / (1 + pow(np.e, -(W1 @ x_grad + b1)))
172         sig_z_grad = 1 / (1 + pow(np.e, -(W2 @ sig_y_grad + b2)))
173         sig_c_grad = 1 / (1 + pow(np.e, -(W3 @ sig_z_grad + b3)))
174         predicted_number = np.where(sig_c_grad == np.amax(sig_c_grad))
175         real_number = np.where(train_data[1] == np.amax(train_data[1]))
176         if predicted_number == real_number:
177             counter += 1
178     print('Accuracy:', counter / number_of_train)

```

نمودار این قسمت به ازای accuracy =65% به صورت زیر است:



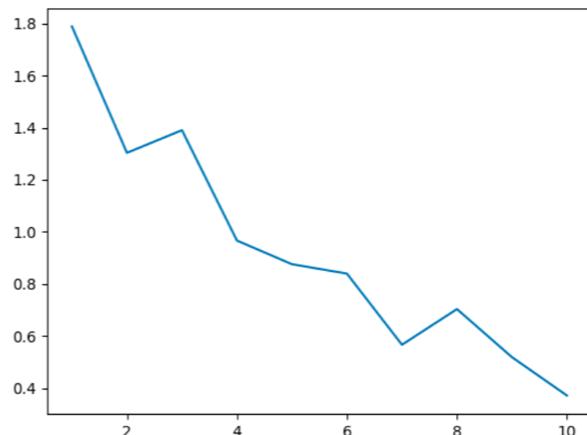
#### قدم چهارم:

مانند قدم اول است تنها با این تفاوت که به جای `for loop` از `matrix` استفاده میکنیم تا مدت اجرای برنامه کمتر شود(از چیزی حدود ۲ دقیقه به کمتر از ۳۰ ثانیه برسد)

```
95
96     for k in range_(batch_num):      #20
97         image = train_set1[batch * 20 + k][0]
98         label_grad = train_set1[batch * 20 + k][1]
99         sig_y_grad = 1 / (1 + pow(np.e, -(W1 @ image + b1)))
100        sig_z_grad = 1 / (1 + pow(np.e, -(W2 @ sig_y_grad + b2)))
101        sig_c_grad = 1 / (1 + pow(np.e, -(W3 @ sig_z_grad + b3)))
102
103        W3_grad += (2 * (sig_c_grad - label_grad) * sig_c_grad * (1 - sig_c_grad)) @ np.transpose(sig_z_grad)
104
105        # bias
106        b3_grad += 2 * (sig_c_grad - label_grad) * sig_c_grad * (1 - sig_c_grad)
107
108        #3rd layer
109        delta_3 = np.zeros((60, 1))
110        delta_3 += np.transpose(W3) @ (2 * (sig_c_grad - label_grad) * (sig_c_grad * (1 - sig_c_grad)))
111
112        # weight
113        W2_grad += (sig_z_grad * (1 - sig_z_grad) * delta_3) @ np.transpose(sig_y_grad)
114
115        # bias
116        b2_grad += delta_3 * sig_z_grad * (1 - sig_z_grad)
117
118        #2nd layer
119        delta_2 = np.zeros((150, 1))
120        delta_2 += np.transpose(W2) @ (delta_3 * sig_z_grad * (1 - sig_z_grad))
121
122        # weight
123        W1_grad += (delta_2 * sig_y_grad * (1 - sig_y_grad)) @ np.transpose(image)
124
125        # bias
126        b1_grad += delta_2 * sig_y_grad * (1 - sig_y_grad)
127
128        # upgrade the weights and layers
129        W3 = W3 - (learning_rate * (W3_grad / batch_size))
```

Looks like you're using NumPy  
Would you like to turn scientific mode on?  
Use scientific mode Keep current layout...

نمودار این قسمت به ازای  $accuracy = 72\%$  به صورت زیر است:



### قدم پنجم:

مانند همان قدم چهارم است اما اینجا پس از train دادن شبکه عصبی آن را test میکنیم که در این حالت accuracy و train هر دو حدود 90% است.

```
153 # find the accuracy of train
154 counter = 0
155 for train_data in train_set[:number_of_train]:
156     x_grad = train_data[0]
157     sig_y_grad = 1 / (1 + pow(np.e, -(W1 @ x_grad + b1)))
158     sig_z_grad = 1 / (1 + pow(np.e, -(W2 @ sig_y_grad + b2)))
159     sig_c_grad = 1 / (1 + pow(np.e, -(W3 @ sig_z_grad + b3)))
160     predicted_number = np.where(sig_c_grad == np.amax(sig_c_grad))
161     real_number = np.where(train_data[1] == np.amax(train_data[1]))
162     if predicted_number == real_number:
163         counter += 1
164 print('Accuracy of train:', counter / number_of_train)
165
166
167 # find the accuracy of test
168 counter2 = 0
169 for test_data in test_set:
170     x_grad = test_data[0]
171     sig_y_grad = 1 / (1 + pow(np.e, -(W1 @ x_grad + b1)))
172     sig_z_grad = 1 / (1 + pow(np.e, -(W2 @ sig_y_grad + b2)))
173     sig_c_grad = 1 / (1 + pow(np.e, -(W3 @ sig_z_grad + b3)))
174     predicted_number = np.where(sig_c_grad == np.amax(sig_c_grad))
175     real_number = np.where(test_data[1] == np.amax(test_data[1]))
176     if predicted_number == real_number:
177         counter2 += 1
178 print('Accuracy of test:', counter2 / number_of_test)
179
```

نمودار این قسمت به از ای accuracy = 98% به صورت زیر است:

