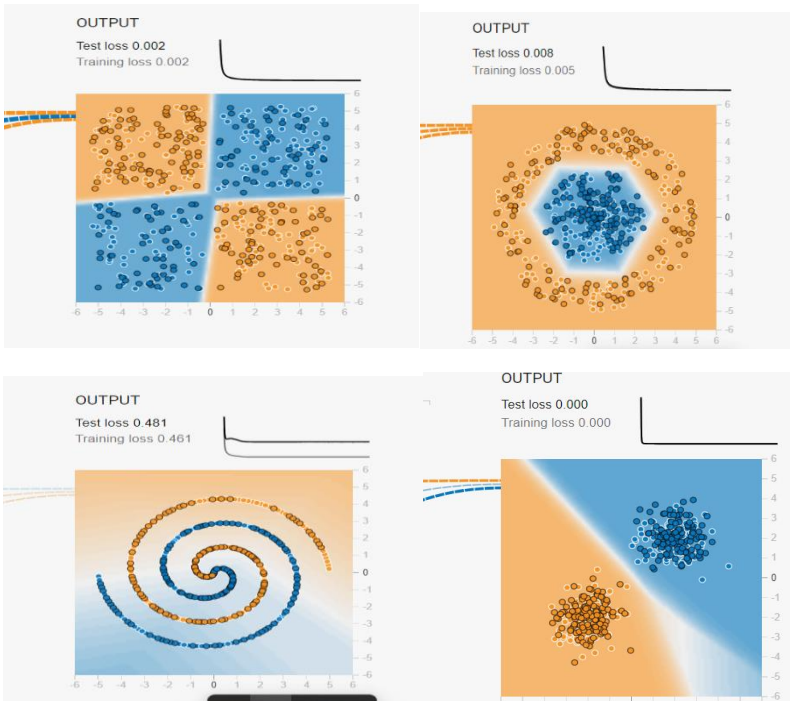
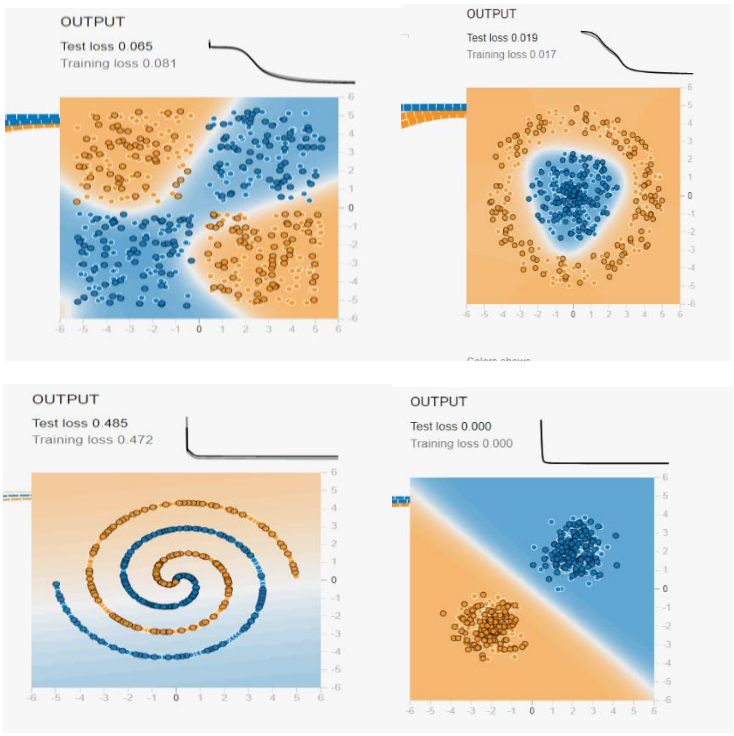
شماره سوال 3-4-5:

سوال 3) مشاهده نتایج اولیه با یک لایه پنهان با 3 نورون و 500 مرحله اموزش

RelU Function

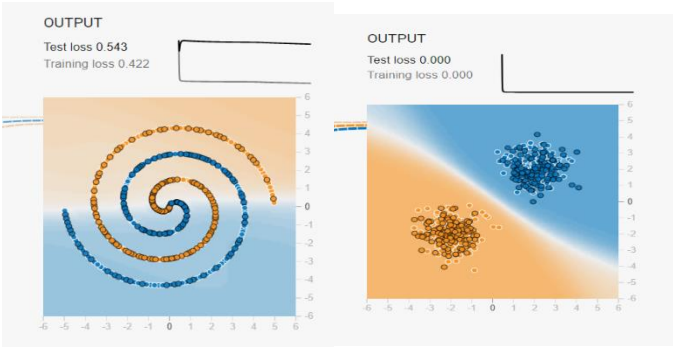


Sigmoid Function

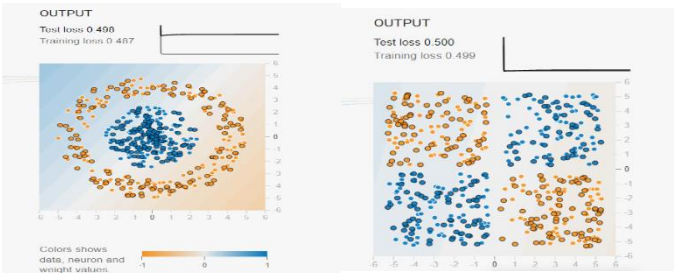


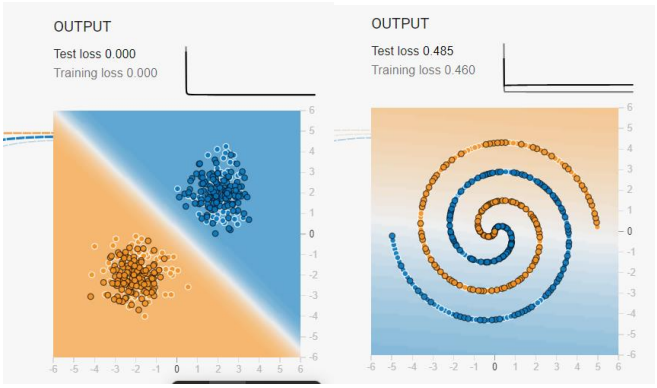
TanH


OUTPUT
Test loss 0.142
Training loss 0.139
OUTPUT
Test loss 0.013
Training loss 0.007


OUTPUT
Test loss 0.543
Training loss 0.422
OUTPUT
Test loss 0.000
Training loss 0.000

Linear


OUTPUT
Test loss 0.498
Training loss 0.487
OUTPUT
Test loss 0.500
Training loss 0.499
Colors shows data, neuron and weight values.


OUTPUT
Test loss 0.000
Training loss 0.000
OUTPUT
Test loss 0.485
Training loss 0.460

-مقایسه چهار تابع فعال سازی بر روی چهار دیتای مختلف:

RelU :

همان طور که مشخص است، در میان تمامی توابع فعال سازی نتیجه ی دقیق تر و خطای کم تری را داراست.

کلسیفیکیشن در هر چهار حالت دقیق است و دیتاست اخر(حالت مارپیچی) نتیجه ی ضعیف تری را داراست اما نتیجه آن بهتر از بقیه ی تابع ها است.

Sigmoid :

همان طور که مشخص است، این تابع فعال سازی به نسبت نتیجه خوبی را گرفته است. بهترین نتیجه این تابع در دیتا ست سوم(به صورت خطی قابل انجام است) و دیتا ست اول است. دیتا ست دوم به نسبت RelU ,ضعیف تر عمل کرده است اما از TanH نتیجه بهتری دارد. دیتا ست چهارم نتیجه دقیق و خوبی ندارد.

TanH :

به صورت کلی به نسبت بقیه فعال ساز ها به جز RelU نتیجه خوبی دارد. به جز دیتا ست خطی که تمام فعال ساز ها روی آن خوب عمل کرده اند، بر روی دیتا ست اول بهتر از بقیه دیتا ست ها عمل کرده است.بر روی دیتا ست دوم نسبت به , sigmoid relu نتیجه ضعیف تری دارد و در دیتا ست چهارم بدترین عملکرد را داراست.

Linear : همان طور که مشخص است، این فعال ساز تنها رو داده های خطی خوب عمل می کند و همان طور که در نتیجه ها مشخص است، تنها نتیجه خوب آن بر روی دیتا ست سوم است که دقیق تشخیص داده است و در بقیه موارد با اختلاف زیاد با بقیه توابع غیر خطی فاصله دارد.

سوال 4)

تعریف یک مدل ساده تنسور فلو :

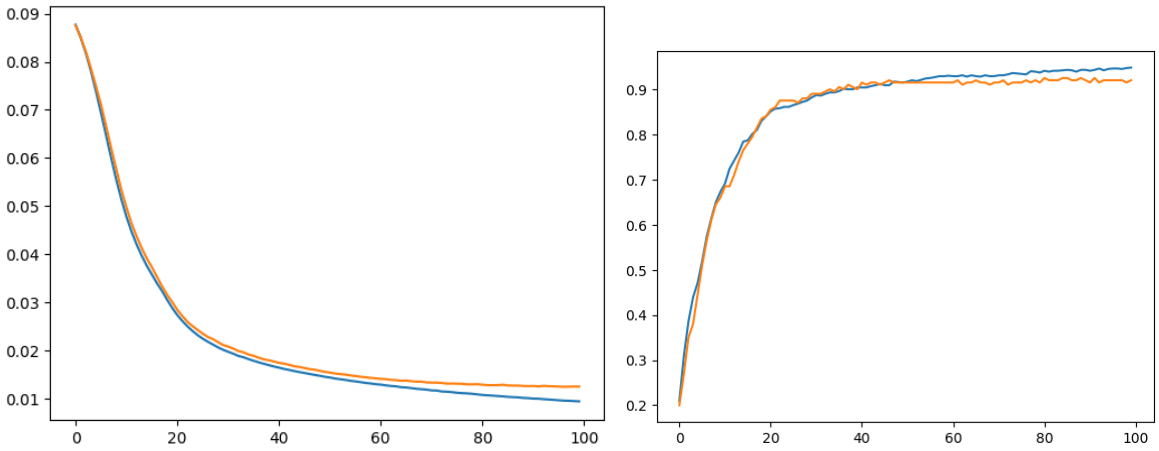
```
model = Sequential([
    Dense(25, input_shape=(25,), activation='relu'),
    Dense(10, activation='softmax')
])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 25)                650

 dense_1 (Dense)             (None, 10)                260

=================================================================
Total params: 910 (3.55 KB)
Trainable params: 910 (3.55 KB)
Non-trainable params: 0 (0.00 Byte)
```

کامپایل مدل با اپتیمایزر adamو loss mean squared root (MSE)

```
#################################
model.compile(optimizer="adam", loss="mean_squared_error",
metrics=["accuracy"])
#################################
```

نتیجه :

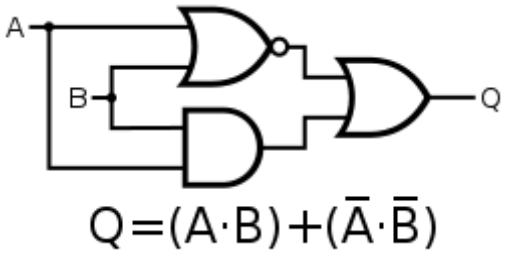


نتیجه سمت چپ مربوط به نمودار خطا است که مشاهده می شود با جلو رفتن به مرور خطای مدل کم تر می شود. در

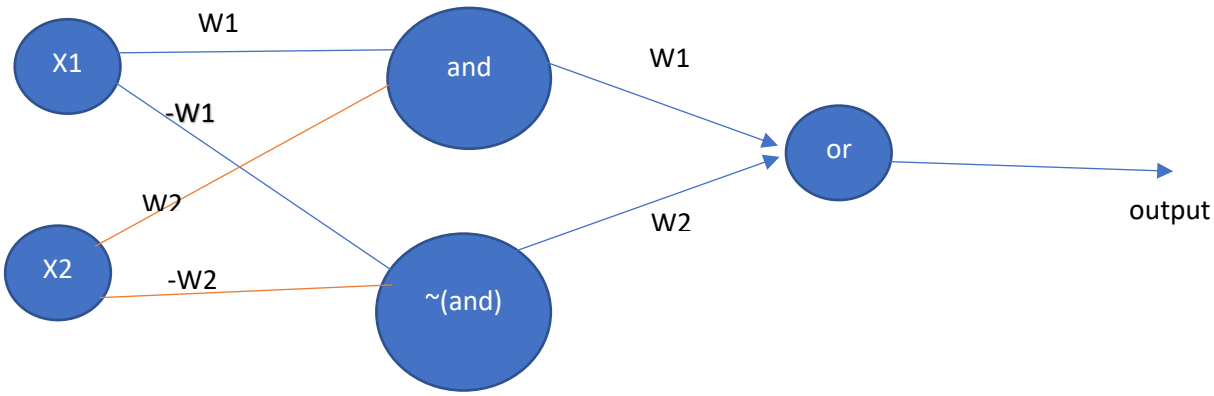
برای accuracy، همان طور که مشخص است بیشتر می شود. و به این معنی است که مدل ما در حال اموزش است و با گذر زمان accurayبیشتر و خطای کم تری دارد. همان طور که مشخص است به مقدار خیلی کمی نتایج داده ولیدیشن با اصلی متفاوت است و lossبرای داده ی ولیدیشن در اواخر دوره ی اموزش بیشتر از داده ی ترین می باشد.

سوال 5)

برای اموزش Xnorاز سه نورون استفاده می کنیم.



برای این مدل ما 3 نورون تعریف می کنیم. نورون اول تعریف کننده ی andماست. نورون دوم نیز اند را تعریف می کند با این تفاوت که به علت نات بودن ورودی ها، در مدل نیز نات ورودی ها به نورون دوم داده می شود. در نورون سوم که به عنوان نورون اور تعریف میشود، نتیجه نهایی است. w1برای x1، w2برای x2 است و مدل حدودی به شکل زیر است.



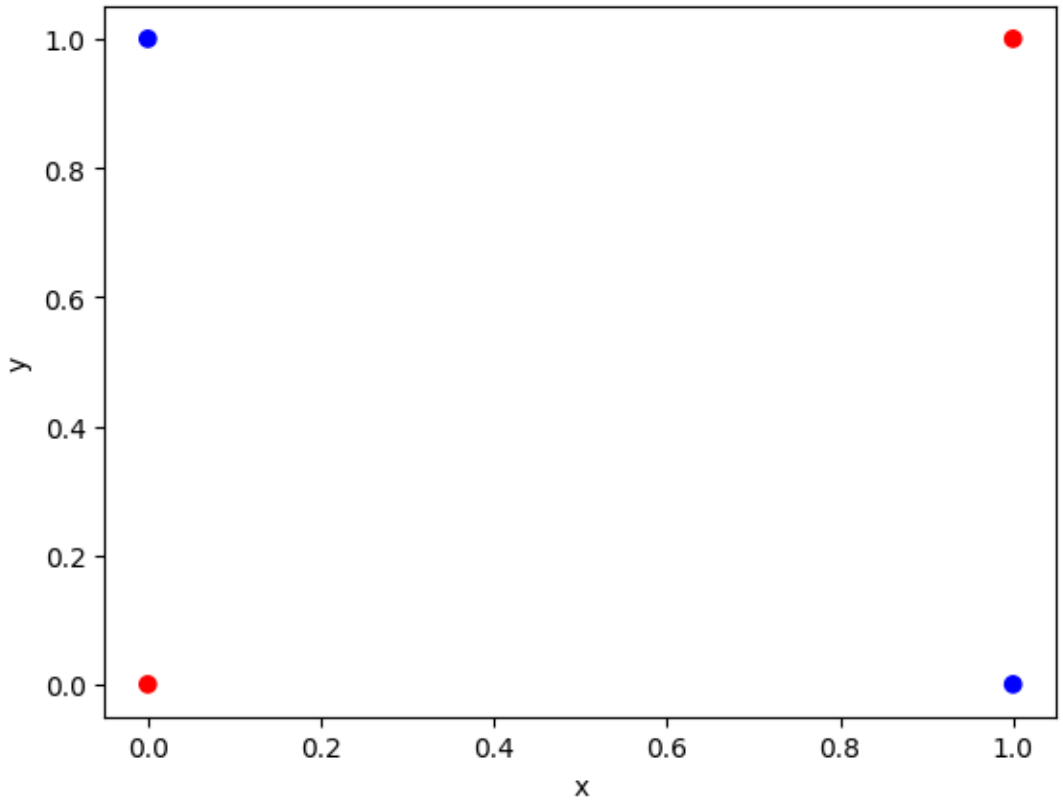
حال بخش به بخش کدمان را توضیح میدهیم.

دیتای سمپل:

```
#dataset and label
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y =np.array([1, 0, 0, 1])
```

نشان دادن نقاط خواسته شده به صورت رنگی

```
colors = ['red' if label == 1 else 'blue' for label in y]
plt.scatter(x[:, 0], x[:, 1] , c=colors)
plt.xlabel('x')
plt.ylabel('y')
```



ابتدا تابع فعال ساز و تابع محاسبه گر خطا رابه صورت ساده تعریف می کنیم

```
class MLP:
    def __init__(self, epochs, learning_rate):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.random.randn(3)
def step_func(z):
    return 1 if (z > 0) else 0

def loss(result, x):
    return result - x
```

کد زده شده، یک کلاس است که learning rate , epochsرا می گیرد و از 2 نورون میانی تشکیل شده است.

نورون اول برای and است و نورون دوم برای Norیا نات ورودی های اند و نورون سوم اور. مقادیر وزن ها هم مقادیر رندم بین 0-1 تعریف می کنیم

```
class MLP:
    def __init__(self, epochs, learning_rate):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.random.randn(3)
```

حالا طبق یک مدل ساده ، forward propagationرا تعریف میکنیم.

$$\sum wi.xi \ + b$$

```
    def forward(self, x1, x2):
        # First node: AND
        self.and_node = step_func(self.weights[0] + self.weights[1]*x1 +
self.weights[2]*x2)
        # Second node: NOR
        self.nor_node = step_func(self.weights[0] + self.weights[1]*(1-x1)
+ self.weights[2]*(1-x2))
        # Final node: OR
        self.output = step_func(self.weights[0] +
self.weights[1]*self.and_node + self.weights[2]*self.nor_node)
        return self.output
```

در اینجا چون نورون دوم طبق فرمول ورودی هایش نات aاست ، نات آن را به ورودی می دهیم.

حال مقادیر وزن ها را با کمک فرمول gradiant descentطبق اسلاید ها تعریف می کنیم.

```
   def update(self, x1, x2, result):
        # Calculate loss
        current_loss = loss(result, self.output)
        # Update weights
        self.weights[0] += self.learning_rate * current_loss
        self.weights[1] += self.learning_rate * current_loss *
self.and_node
        self.weights[2] += self.learning_rate * current_loss
*  self.nor_node
```

حالا داده را ترین می کنیم. در ترین یک بار فرورارد و یکبار اپدیت را صدا می زنیم.

```
    def train(self, x_train, y_labels):
        for epoch in range(self.epochs):
            i = 0
            for data in x_train:
                x1 = data[0]
                x2 = data[1]
                result = y_labels[i]
                self.forward(x1, x2)
                self.update(x1, x2, result)
                i += 1
```

حال نتیجه را با , learning_rate 0.1 , و دوره ی 150 می بینیم.

برای دیدن نتیجه حاصل با کمک مدل زده شده :

```
    def predict(self, x1, x2):
        return self.forward(x1, x2)
```

حال نتیجه را ترین میکنیم .

```
mlp = MLP(150, 0.1)
mlp.train(x, y)

# Print final weights
print("Final Weights:")
print(mlp.weights)
i = 0
for data in x:

    x1 = data[0]
    x2 = data[1]
    prediction = mlp.predict(x1, x2)
    print(f"Input: ({x1}, {x2}), Label: {y[i]}, Prediction: {prediction}")
    i+=1
```

```
Final Weights:
[ 0.08711273 -0.04193251 -0.04792214]
Input: (0, 0), Label: 1, Prediction: 1
Input: (0, 1), Label: 0, Prediction: 0
Input: (1, 0), Label: 0, Prediction: 0
Input: (1, 1), Label: 1, Prediction: 1
```

همان طور که مشخص است، نتیجه پیش بینی شده مطابق با سوال است.

سوال 6)

این دیتا ست شامل عکس های 28*28 ، و سیاه و سفید است. ابتدا یک مدل سیکونشوال تعریف می کنیم که ورودی ان به اندازه 784 ، (برای ورودی دیتا را با کمک flattenابعاد آن را یک بعدی می کنیم و با تقسیم بر 255 نرمالایز می کنیم.

لایه های میانی شامل یک لایه میانی 256 نورون است.. در انتها 10 نورون که هر کدام برای هر کدام از اعداد صفر تا 9 است. اپتیمایزر آن adamو loss functionآن نیز cross antropyاست چرا که میخواهد به دسته های مختلف classifyکند.

اکتیویت فانکشن لایه اخر نیز برای عملکرد بهتر و دسته بندی به گروه های مختلف softmax است.

برای داده ها ، دسته بندی و نرمال کردن آن ها :

```
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))
train_images = train_images / 255.
test_images = test_images.astype("float32") / 255
```

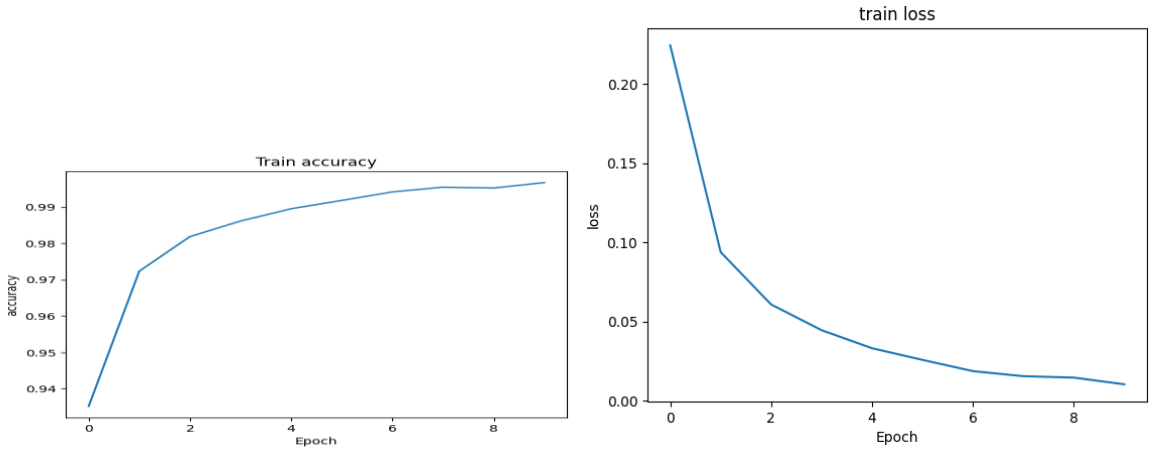
مدل تعریف شده :

```
model = keras.models.Sequential([
    keras.layers.Input(shape=(None, 784)),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

کامپایل متد:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logi
ts=True),
              metrics=['accuracy'])
```

نتیجه ی فانکشن لاس و اکیوریسی :



همان طور که دیده می شود اکیوریسی نیز بالای 95 درصد است.

```
313/313 - 1s - loss: 0.0769 - accuracy: 0.9817 - 691ms/epoch - 2ms/step
[0.0769181177020073, 0.9817000031471252]
```