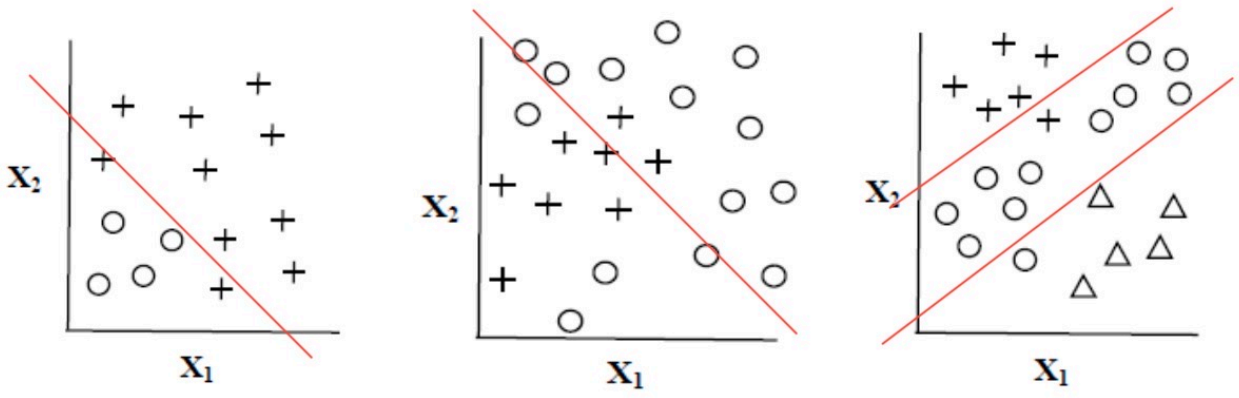
سوال ۱

الف) این الگوریتم دارای دو مرحله است؛ که **مرحله اول** حرکت به جلو یا feed forward نامیده می‌شود. ابتدا داده‌های ورودی را در وزن‌ها ضرب و سپس با انحراف یا همان بایاس جمع می‌کنیم، برای هر نورون مقدار بدست‌آمده را از یک تابع انگیزش یا activation function عبور می دهیم؛ برای هر لایه این عمل تکرار می‌شود تا در آخر به یک خروجی می رسیم که به احتمال زیاد با خروجی واقعی تفاوت دارد در اینجا با تابع هزینه یا cost function میزان خطا را مشخص می‌کنیم، پس از آن به **مرحله دوم** در یک تکرار می‌رویم. در این مرحله می‌توانیم به عقب بازگشته و وزن‌ها و انحراف‌ها را به‌هنگام‌سازی کنیم به این معنا که وزن‌ها را به گونه‌ای تغییر دهیم که در تکرارِ بعدی با خطای کمتری مواجه شویم. برای انجام اینکار ابتدا از تابع خطا مشتق گرفته تا بتوانیم گرادیان را محاسبه کنیم و عمل پس‌انتشار را انجام دهیم. با محاسبه گرادیان مقدار تغییر وزن هر نورون، بسته به اینکه در کدام لایه قرار دارد، را بدست می‌آوریم. این تکرار تا زمانیکه به نزدیک‌ترین مقدار واقعی برای خروجی برسیم، ادامه پیدا می‌کند.

ب) روش پس‌انتشار حل مسائل چند لایه را امکان پذیر می‌کند؛ به این صورت که با استفاده از این روش می‌توانیم در هر نورون یک ویژگی را به گونه‌ای یاد بگیریم که انگار توسط متخصصان انسانی طراحی شده است. به دلیل کاراییِ الگوریتم و این امر که دیگر نیازی به متخصصان انسانی نبود، این روش در حل مسائل پیچیده، که نیاز به زمان و هزینه‌ی بالاست، کاربرد وسیعی دارد. به صورت تخصصی می‌توان گفت این روش باعث می‌شود که با تغییر وزن‌ها، رفته‌رفته خطایِ پیش‌بینی کاهش پیدا کند.

سوال ۲

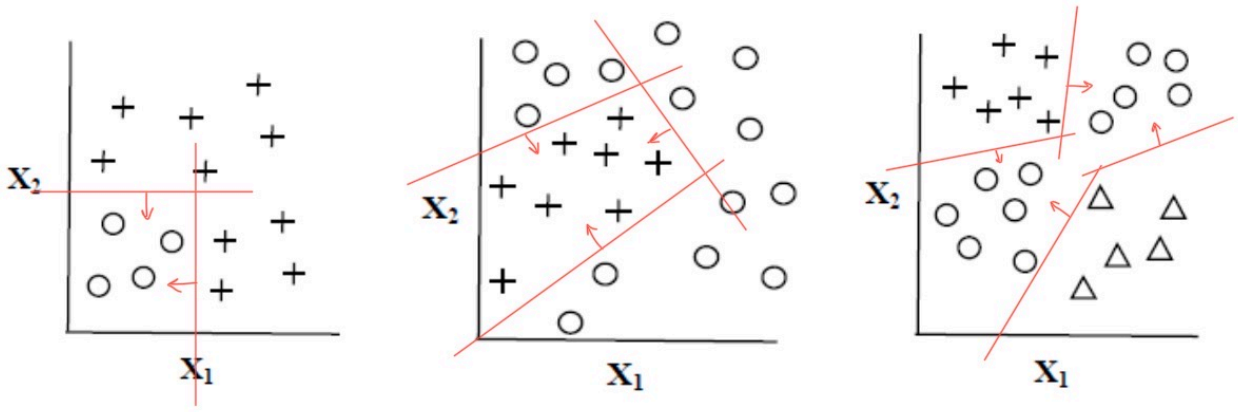
الف)



(الف) (ب) (پ)

کم دقت‌ترین:

در شکل الف بدلیل اینکه داده‌ها دارای توزیع مناسبی هستند با تنها یک خط جداسازی نسبتا به خوبی انجام شده است، اما در شکل ب توزیع داده‌ها ناموزون است و دارای پراکندگی می‌باشد، بنابراین جداسازی با تنها یک خط، خطای بالایی را به ما می‌دهد. شکل پ بدلیل وجود سه نوع داده مسئله با دو خط جدا شده است و نیاز است که لایه‌های دیگر اضافه شوند تا بتوانیم عمل جداسازی را بهتر انجام دهیم.



(الف) (ب) (پ)

با دقت‌ترین:

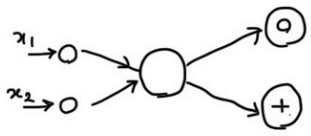
حال با اضافه کردن لایه، می‌توانیم از چندین خط برای جداسازی استفاده کنیم. در شکل الف اگر بخواهیم درصد پایینی از خطا را داشته باشیم از دو خط استفاده می‌کنیم؛ برای شکل ب برای ایجاد دقت مناسب از سه خط استفاده شده اما بدلیل اینکه شکل الف توزیع داده مناسبی دارد این عمل سریعتر صورت می‌پذیرد و تغییر در میزان دقت، نسبت به حالتی که از یک خط برای جداسازی استفاده کردیم، کمتر است. شکل پ با اجتماع چهار خط جداسازی می‌شود و نسبت به دو شکل دیگر به آموزش طولانی‌تری نیاز دارد چون مسئله‌ی پیچیده‌تری است.

ب)

پ) زمانیکه که مسئله‌ی ما به صورت خطی جداپذیر نباشد یا گفته می‌شود که مسئله پیچیده شده است، از جداسازهای غیر خطی استفاده می‌کنیم.

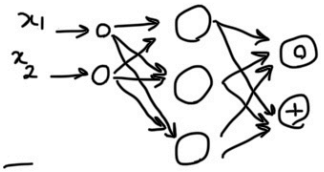
د)

الف)



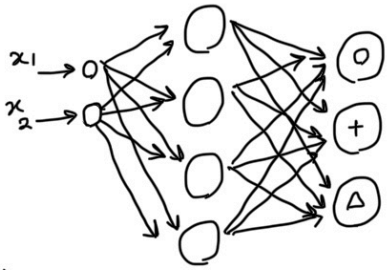
از آنجایی که دو ورودی داریم در لایه اول دو نورون وجود دارد و چون مسئله ی ما با یک خط نیز به خوبی جدا می شود از یک نورون در لایه بعد به عنوان جداساز استفاده کرده ایم و در آخر شبکه ما در صدد تعلق به هرکدام از کلاس های ما را می دهد که در اینجا دو کلاس تحت عنوان o و + وجود دارد پس لایه خروجی ما دارای دو نورون می باشد.

ب)



در این شبکه همانند بالا دو ورودی و دو خروجی داریم اما در لایه وسط از سه نورون استفاده کردیم چون مسئله ما با سه خط قابل جداسازی است و هرکدام از این نورون ها معادله یکی از خطوط را به ما می دهد.

پ)



در شکل سوم مانند دو مورد قبل دو ورودی داریم اما چون برای جداسازی آن ها باید از چهار خط استفاده کنیم در لایه میانی چهار نورون قرار می دهیم و به دلیل وجود سه نوع کلاس برای خروجی در لایه output یا آخر سه نورون می گذاریم تا تعلق به هرکدام بررسی شود.

سوال ۳

فایل jpg ضمیمه شده است.

سوال ۴

الف) کد مربوطه ضمیمه شده است. توضیحات مربوط به هربخش در کد به صورت کامنت ذکر شده است.

تهیه کننده: فاطمه طاهر

ب) از آنجایی که خروجی مد نظر ما ۰ یا ۱ می‌باشد، تابع انگیزش سیگموئید گزینه‌ی مناسب‌تری نسبت به تابع خطی است و با استفاده از آن سریع‌تر به جواب مطلوب خود می‌رسیم؛ از طرف دیگر چون این تابع نسبت به تابع خطی پیچیده‌تر است و محاسبات بیشتری را می‌طلبد زمانی که خروجی را برای ما تولید می‌کند نیز بیشتر می‌باشد.

پ)

- **اضافه کردن نورون‌های لایه‌ی پنهان:** هر چه تعداد این نورون‌ها افزایش پیدا کند، ما می‌توانیم ویژگی‌ها را دقیق‌تر استخراج کنیم. وزن مرتبط به نورون، تاثیر این ویژگی در تعیین دسته‌بندی آخر را مشخص می‌کند.
- **تعداد لایه‌های پنهان:** هرچه پیچیدگی داده‌های ورودی بیشتر باشد برای آموزش به لایه‌های بیشتری نیاز است؛ البته پیچیده‌ترین مسئله‌ها با سه لایه مخفی قابل حل هستند و در صورت وجود لایه‌های بیشتر مدل overfit می‌شود. که در اینجا چون داده‌ها پیچیدگی خاصی ندارند و هر بردار ورودی تنها متشکل از سه نورون می‌باشد همان یک لایه هم برای مسئله‌ی ما پاسخگوست.
- **نرخ یادگیری:** نرخ یادگیری، اندازه گام را برای محاسبه گرادیان مشخص می‌کند. در ابتدا بهتر است این عدد را میزانی نزدیک به یک در نظر بگیریم و سپس رفته رفته آن را کاهش دهیم. اگر میزان آن بیش از اندازه بزرگ باشد نقطه مینیمم محلی را از دست می‌دهیم و اگر بسیار کوچک باشد، گام‌های ما به سمت مینیمم بسیار کوچک هستند و به کندی به نقطه مینیمم می‌رسیم. برای این داده‌ها طبق نتایجی که از تست بدست آمده است عدد ۰/۱ مناسب می‌باشد.

ت) در صورتیکه نرخ یادگیری ثابت باشد، سیر پیشرفت ما از همان ابتدا با گام‌های کوچک و کند طی می‌شود و اگر از نقطه بهینه‌مان فاصله‌ی زیادی داشته باشیم نیاز داریم تا از داده‌های آموزشی بیشتری استفاده کنیم. با استفاده کردن از نرخ یادگیری وفقی عمل سیر کردن به نقطه بهینه سریعتر صورت می‌گیرد و احتمال اینکه در نقاط بهینه محلی گیر کاشه می‌یابد، سرعت آموزش نیز بالاتر می‌رود.

سوال ۵

برای لود کردن دیتاست mnist از دستور زیر استفاده می‌کنیم:

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

با این دستور دیتاست ما که دارای ۷۰ هزار نمونه‌ی آموزشی می‌باشد به دو دسته آموزش و تست تقسیم می‌شود که ۶۰ هزار از تصاویر برای آموزش و مابقی برای تست در نظر گرفته

می‌شوند. هرکدام از تصاویر دارای اندازه ۲۸ در ۲۸ هستند و یک لیبل، که معرف عدد موجود در تصویر می‌باشد.

در مرحله بعد بخش آموزشی را به دو قسمت validation و train تقسیم می‌کنیم.:

```
x_val  = x_train[50000:60000]
x_train = x_train[0:50000]
y_val  = y_train[50000:60000]
y_train = y_train[0:50000]
```

اینکار بیش‌برازش را در مراحل ابتدایی تشخیص می‌دهد و از آن جلوگیری می‌کند.

برای اینکه در لایه‌ی ابتدایی تمامی پیکسل‌های تصویر دیده شوند از reshape کردن استفاده می‌کنیم:

```
x_train = x_train.reshape(50000, 784)
x_val = x_val.reshape(10000, 784)
x_test = x_test.reshape(10000, 784)
```

با اینکار لایه‌ی ورودیِ ما دارای ۷۸۴ نورون می‌شود که هرکدام مقدار بین ۰ تا ۲۵۵ (سطوح خاکستری در تصویر) دارند.

برای اینکه عمل آموزش سریعتر صورت پذیرد و به دقت بالاتری برسیم، از نرمال‌سازی داده ورودی استفاده میکنیم. در اینجا بازه سطوح خاکستری را به جای ۰ تا ۲۵۵ به ۰ تا ۱ نگاشت کرده‌ایم:

```
x_train = x_train.astype('float32')
x_val = x_val.astype('float32')
x_test = x_test.astype('float32')
gray_scale = 255
x_train /= gray_scale
x_val /= gray_scale
x_test /= gray_scale
```

در لایه‌ی آخر ۱۰ کلاس متناظر با اعداد ۰ تا ۹ وجود دارد، پس خروجی ما برای تشخیص عدد باید آرایه‌ای ۱۰ در ۱ باید که تنها یکی از درایه‌های آن ۱ و بقیه صفر می‌باشند. تبدیل برچسب‌ها در این دیتاست با one hot encoding به صورت زیر انجام می‌شود:

```
num_classes = 10
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_val = tf.keras.utils.to_categorical(y_val, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

تخصیص لایه‌های ورودی و خروجی به شکل زیر صورت می‌پذیرد:

```
x = tf.placeholder(tf.float32, [None, 784])
```

```
y = tf.placeholder(tf.float32, [None, 10])
```

برای آموزش از سه لایه استفاده شده است که هر نورون در لایه اول دارای ۷۸۴ ورودی (به اندازه تعداد ورودی‌ها) می‌باشد. لایه اول دارای ۲۵۶ نورون و لایه دوم ۱۲۸ نورون دارد که تابع برازش برای هردو relu می‌باشد. در لایه سوم ما ۱۰ نورون داریم که از softmax برای برازش آن استفاده می‌کنیم که تعداد نورون‌های آن متناظر با تعداد کلاس‌ها در خروجی است. (تابع mlp در کد اینکارها را برای ما انجام می‌دهد.)

فراخوانی تابع:

```
logits = mlp(x)
```

در اینجا ورودی‌ها در وزن‌ها ضرب می‌شوند و در شبکه به سمت جلو حرکت می‌کنند. برای اینکه عملکرد مدل را روی نمونه دیده‌نشده ارزیابی کنیم از sofmax_cross_entropy استفاده می‌کنیم (از Adam هم برای بهینه کردن میزان کاهش یا loss استفاده شده است.):

```
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
    logits=logits, labels=y))
```

```
train_op = tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss_op)
```

برای استفاده از tensorflow ابتدا باید آن را پایه‌گذاری کنیم:

```
init = tf.global_variables_initializer()
```

حال پارامترهایِ لازم مانند تعداد دوره، اندازه دسته و همینطور با توجه به این پارامتر تعداد تکرار در آموزش شبکه را تعیین می‌کنیم:

```
epoch_cnt = 30
batch_size = 1000
iteration = len(x_train) // batch_size
```

از آنجایی که ما ۳۰ ایپوک داریم، ۳۰ عدد برای validation نیز بدست می‌آوریم. با تعیین اندازه دسته روی ۱۰۰۰ تعداد تکرارهای ما ۵۰ بار می‌شود و میانگینی از میزان loss را بدست می‌آوریم:

```
for i in range(iteration):
        _, loss = sess.run([train_op, loss_op],
                           feed_dict={x: x_train[start: end], y: y_train[start:
end]})
        start += batch_size; end += batch_size
        # Compute average loss
        avg_loss += loss / iteration
```

همانطور که توضیح داده شد با استفاده از softmax در لایه‌ی آخر، کلاس مربوط به تصویر داده شده را توسط مدل پیش‌بینی می‌کنیم:

```
preds = tf.nn.softmax(logits)
correct_prediction = tf.equal(tf.argmax(preds, 1), tf.argmax(y, 1))
```

بعد میزان صحت آن را با برچسبی که به آن تخصیص داده شده بود، میسنجیم:

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
cur_val_acc = accuracy.eval({x: x_val, y: y_val})
```

بعد از پایان یافتن دوره‌ها (در اینجا ۳۰ دوره) با داده دیده‌نشده تحت عنوان تست، شبکه را مورد آزمون قرار می‌دهیم:

```
preds = tf.nn.softmax(logits)  # Apply softmax to logits
    correct_prediction = tf.equal(tf.argmax(preds, 1), tf.argmax(y, 1))
```

و در آخر دقت سیستم را روی داده‌های تست بدست می‌آوریم:

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

(این کد روی colab اجرا شده است و فایل ضمیمه آن هم به فرمت py. و هم ipynb. ارسال شده است.)