دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه‌ی اول هوش محاسباتی

MNIST with FC_MLP

پدیدآورنده:

محمد امین کیانی

۴۰۰۳۶۱۳۰۵۲

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد راهنما: جناب اقای دکتر تابع الحجه

نیمسال دوم تحصیلی ۰۳-۱۴۰۲

فهرست مطالب

مستندات

۱-مسئله و تحلیل کلی آن:

از جستجوی بصری برای بهبود قابلیت کشف محصول تا تشخیص چهره در شبکه‌های اجتماعی - طبقه‌بندی تصاویر به یک انقلاب بصری آنلاین دامن می‌زند و دنیا را طوفانی کرده است. طبقه بندی تصویر، زیرشاخه بینایی کامپیوتری به پردازش و طبقه بندی اشیا بر اساس الگوریتم های آموزش دیده کمک می کند.طبقه‌بندی تصویر لحظه اورکا خود را در سال ۲۰۱۲ داشت، زمانی که Alexnet برنده چالش ImageNet شد و از آن زمان به بعد رشد تصاعدی در این زمینه مشاهده شد. در حالی که ما انسان ها توانایی خود را برای طبقه بندی آسان اشیاء اطراف خود می دانیم زیرا مغز ما به طور ناخودآگاه با مجموعه ای از تصاویر آموزش داده شده است، مشکل به این راحتی ها نیست. عوامل متعددی مانند تغییر دیدگاه، تغییر اندازه، انسداد (ترکیب اشیاء با سایر اشیاء در تصویر)، تفاوت در جهت و منبع نور، طبقه‌بندی صحیح تصاویر را برای ماشین‌ها دشوار می‌کند. با این وجود، این یک زمینه هیجان انگیز و رو به رشد است و راهی بهتر از طبقه بندی تصاویر در مجموعه داده MNIST برای یادگیری اصول اولیه طبقه بندی تصاویر وجود ندارد.

۲-تاثیر نرمالایز کردن:

```
# Reshape and normalize the images.
X_train = train_images.reshape((60000, 784))
X_train = X_train.astype('float32') / 255
X_test = test_images.reshape((10000, 784))
X_test = X_test.astype('float32') / 255
```

با **نرمالایز کردن** داده‌ها در کد و تقسیم آنها بر ۲۵۵، داده‌ها را از مقادیر پیکسل اولیه (بین ۰ تا ۲۵۵) به مقادیر واحد (بین ۰ و ۱) تبدیل می‌کنیم. این مرحله موارد زیر را کمک می‌کند:

۱. استقرار سریع‌تر – اگر برای همگرایی بهتر الگوریتم‌های بهینه‌سازی مانند Stochastic Gradient Descent (SGD)وجود داشته باشد .

۲. جلوگیری از اشباع شدن – اگر از توابع فعال‌سازی مانند Sigmoid یا Tanh استفاده کنیم، این تحول می‌تواند از شبکه‌ی عصبی از اینکه در لایه‌های عمیق به اشباع شود جلوگیری کند.

۳. ایجاد شرایطی بهتر برای یادگیری – این تغییرات معمولاً موجب بهبود همگرایی شبکه و بهبود عملکرد آن می‌شود.

پس با نرمالایز کردن داده‌ها، امکان افزایش سرعت و دقت آموزش شبکه عصبی MLP بر روی دیتاست MNIST وجود دارد. این مرحله تضمین می کند که مدل در طول آموزش سریعتر همگرا می شود و از مسائل مربوط به داده های ورودی در مقیاس های مختلف جلوگیری می کند.

همچنین منظور از **ریشیپ** یعنی تغییر شکل یا ابعاد داده ها با حفظ همان عناصر است. در دیتاست، تصاویر از یک شکل دو بعدی(۲۸x۲۸) پیکسل به یک شکل یک بعدی (۷۸۴ پیکسل) تغییر شکل می دهند. این برای تغذیه داده ها به یک شبکه عصبی پرسپترون چند لایه (MLP) کاملاً متصل ضروری است. زیرا کار با داده ها را در یک شبکه عصبی آسان تر می کند. شبکه عصبی هر پیکسل را به عنوان یک ویژگی ورودی جداگانه در نظر می گیرد.

با تغییر شکل تصاویر از ۲ بعدی به ۱ بعدی، مقادیر پیکسل مسطح شده و در یک ردیف قرار می گیرند. این تبدیل به شبکه عصبی اجازه می دهد تا هر پیکسل را به عنوان یک نورون ورودی جداگانه در نظر بگیرد و آن را برای آموزش MLP کاملاً متصل مناسب می کند.

با **تبدیل برچسب‌ها به فرمت one-hot encoding** (یعنی تبدیل آن‌ها به بردارهای دودویی که هر المان تنها یک پرسشنت از کلاس متناظر را نشان می‌دهد- در روش one-hot encoding، هر برچسب به یک بردار دودویی تبدیل می‌شود که در آن تنها یک عنصر آن برابر با ۱ (پرسشنت) و سایر عناصر بردار برابر با ۰ است. این روش به مدل کمک می‌کند تا بهتر بین دسته‌ها تمایز قائل شود و اطلاعات دقیق‌تری از برچسب‌ها را دریافت کند. اصطلاح "پرسشنت" (One-Hot) اشاره به این دارد که تنها یک عنصر در هر بردار مقدار ۱ دارد که نشان‌دهنده وجود و متعلق بودن به یک دسته خاص است.)، مدل ما قادر خواهد بود بهتر از اطلاعات دقیق برچسب‌ها برای یادگیری و تمایز دادن بین دسته‌ها استفاده کند. من این روش را در درس ماشین لرنینگ نیز برای پردازش رگرسیونی کارت های اعتباری یک دیتاست نیز استفاده کردم و بسیار بهتر از روش مپینگ دستی داده ها و سایر روش هایی که می دانم عمل کرد (در حدود ۴ الی

۶ درصد R2را افزایش و میزان خطارا در حد چندین میلیون کاهش داد – در اینجا نیز به خوبی عمل می کند)

در واقع، این تبدیل باعث افزایش دقت مدل شده و می‌تواند به کاهش مشکلات مربوط به بیش‌برازش (overfitting) کمک کند. این کار معمولاً در برخی از وظایف دسته‌بندی که تعداد دسته‌ها زیاد است و دسته‌بندی دقیقی مهم است (مانند MNIST) مفید است. پس بقیه مقادیر برای هر برچسب به جز آن برابر ۰ است و تنها مقدار مربوط به خود برچسب برابر ۱ است.

به طور کلی، تغییر شکل تصاویر و عادی سازی مقادیر پیکسل، مراحل پیش پردازش مهمی در هنگام کار با داده های تصویر و شبکه های عصبی مانند MLP های کاملاً متصل هستند.

۳-انتخاب مدل و لایه بندی همراه با Dropout و CallBack:

```
# Define the sequential model.
model = tf.keras.models.Sequential([
        # input layer. input data with size 28*28 and output size 256
        # 256 means you set up your modul with 256 NN in this layer.
        tf.keras.layers.Dense(256, input_shape=(28*28,), activation='relu'),

        # hidden layer. input data with size 256, which were same to output of
input layer.
        # output size 256, we set up 256 NN again in this hidden layer.
        # no need to give input size here because keras already know.
        tf.keras.layers.Dense(256, activation='relu'),

        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.BatchNormalization(),  # Add batch normalization layer


        # output layer. the number of output should be your number of
classification
        tf.keras.layers.Dense(10, activation='softmax')
```

```
])
```

یک مدل شبکه عصبی عصر جدید از جنس یک شبکه عصبی مکرر (MLP) که به صورت مکرر (یا "sequentially") لایه‌ها را به یکدیگر وصل می‌کند، تعریف میکنیم. این شبکه معمولا برای پردازش داده‌های ساختار یافته مانند تصاویر استفاده می‌شود. در اینجا، از TensorFlow برای تعریف این معماری استفاده شده است.

سپس نیاز به افزودن لایه‌های لایه‌ها و تنظیمات مدل، مانند تعداد نورون‌ها و توابع فعال‌سازی مربوطه، خواهیم داشت. این مدل می‌تواند در پیش‌بینی یا دسته‌بندی ارقام دست نوشته شده در تصاویر به کار رود.

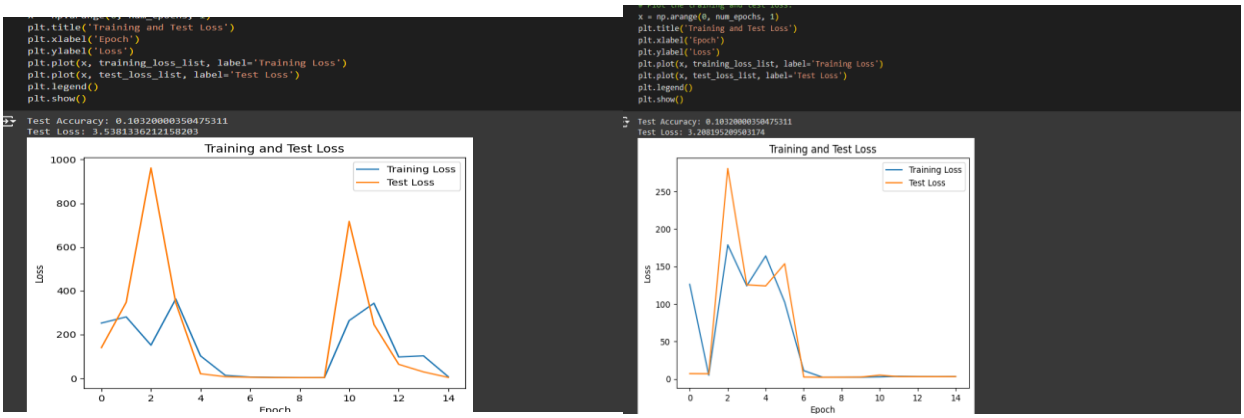
با اضافه کردن دو لایه fully-connected جدید با دوتا ۲۵۶ نرون و ۱۰ نرون به مدل، تعداد پارامترهای قابل آموزش افزایش می‌یابد که این می‌تواند به بهبود عملکرد شبکه کمک کند. با افزایش تعداد لایه‌ها و نرون‌ها، مدل می‌تواند الگوهای پیچیده‌تری را یاد بگیرد و این می‌تواند باعث افزایش دقت مدل در تشخیص اعداد دیجیتال دیتاست MNIST شود. یعنی ابتدا یک لایه fully-connected با ۲۵۶ نورون و تابع فعال‌سازی ReLU اضافه می‌شود. این لایه ورودی ها را به ۲۵۶ نورون هیدن تبدیل می‌کند. سپس در خط بعدی یک لایه fully-connected دیگر با ۱۰ نورون و تابع فعال‌سازی softmax اضافه می‌شود. این لایه ورودی‌های مرحله قبلی را به ۱۰ نورون متناظر با اعداد ۰ تا ۹ تبدیل می‌کند و احتمال تعلق به هر کلاس را بین ۰ و ۱ محاسبه می‌کند.

به طور کلی، اضافه کردن لایه‌های عمیق‌تر می‌تواند به مدل قدرت بیشتری برای یادگیری اطلاعات پیچیده بدهد اما نیاز به مراقبت بیشتر در مورد تطبیق آن با داده‌ها و جلوگیری از overfitting وجود دارد. پس با تست گذاشتن چندین حالت مختلف از تعداد لایه های متفاوت و تحقیق در کتاب هندز ان ماشین لرنینگ میتوان نتیجه گرفت که دو لایه

برای مدل انتخاب شده توسط بنده کافی است و از اورفیت شدن هم جلو گیری می کند و لایه های بیشتر از این نیاز نبوده و تنها سرعت اجرا را پایین می برند و لایه ی کمتر نیز دقت و قدرت را پایین می برد.

اضافه کردن kernel_regularizer=regularizers.l۲(۰.۰۱) برای فیت کردن هرچه بیشتر و دقیق تر عالی بود(رگولاریزیشن L۲ به جمع مجموع مقادیر مربوط به وزن‌ها به عنوان یک جزیی از تابع هزینه مدل اضافه می‌شود، که کمک می‌کند از بیش‌برازش مدل (overfitting) جلوگیری کند با کاهش اهمیت وزن‌های بزرگ.) اما سبب افت دقت و افزایش خطا می شود:

```
tf.keras.layers.Dense(256, activation='relu',
kernel_regularizer=regularizers.l2(0.01)),
```



در بهترین حالت اینگونه بود که روی لایه های نهان اعمال شود ولی باز هم انچنان مفید واقع نشد حتی با تغییر ابر پارامترهایش:

```
model = tf.keras.models.Sequential([
        # input layer. input data with size 28*28 and output size 256
        # 256 means you set up your modul with 256 NN in this layer.
        tf.keras.layers.Dense(256, input_shape=(28*28,),
activation='relu'),
```

```
        # hidden layer. input data with size 256, which were same to
output of input layer.
        # output size 256, we set up 256 NN again in this hidden layer.
        # no need to give input size here because keras already know.
        tf.keras.layers.Dense(256, activation='relu',
kernel_regularizer=regularizers.l2(0.01)),

        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.BatchNormalization(),  # Add batch normalization
layer

        # output layer. the number of output should be your number of
classification
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

tf.keras.layers.BatchNormalization() در TensorFlow یک لایه‌ی Batch Normalization اضافه می‌کند تا آموزش شبکه عصبی سریعتر شود، مشکل محوشوندگی را کاهش می‌دهد، باعث افزایش دقت مدل خود می‌شود و از مشکل برازش بیش‌افرازی جلوگیری می‌کند. این لایه به آموزش شبکه عصبی کمک می‌کند تا به سرعت به یک حالت تعادل مطلوب برسد.

```
tf.keras.layers.Dropout(0.2),
```

این خط کد یک لایه Dropout با نرخ ۰.۲ را به معماری شبکه عصبی اضافه می‌کند. این لایه Dropout به طور تصادفی بخشی از ورودی‌ها را با احتمال ۰.۲ حذف می‌کند. این کار باعث کاهش اورفیت مدل می‌شود و از بروز پدیده‌هایی مانند بیش‌برازش جلوگیری می‌کند.

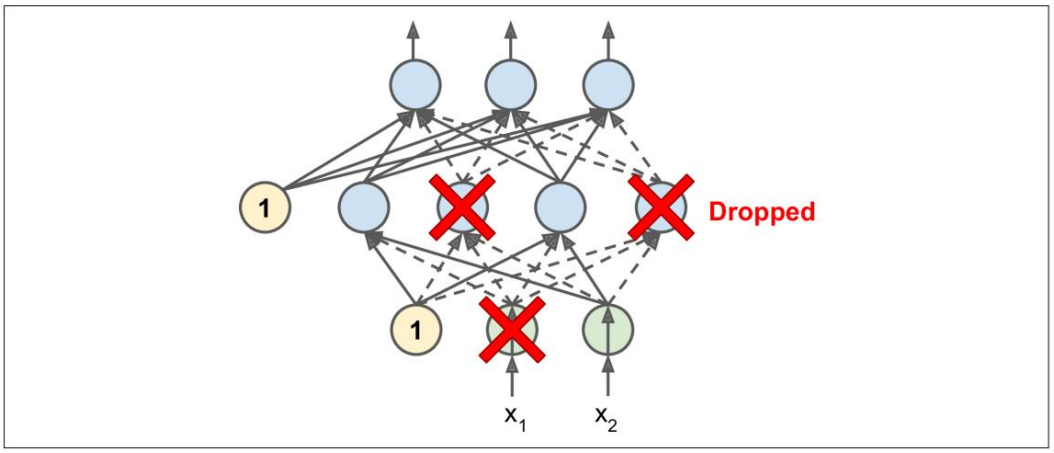


Figure 11-9. Dropout regularization

```
# Define a callback to save the model when validation loss improves.
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='mnist_model.h5',
    save_best_only=True,
    monitor='val_loss',
    mode='min',
    verbose=1
)
```

این کد یک callback به نام ModelCheckpoint برای مدل شبکه عصبی ایجاد می‌کند که:

- filepath='mnist_model.h5': این تنظیم محلی است که مدل آموزش دیده ذخیره می‌شود. مدل به صورت h5 (HDF5) ذخیره می‌شود.

- save_best_only=True: این تنظیم مشخص می‌کند که فقط بهترین نسخه از مدل ذخیره شود (بر اساس مقدار مانیتور شده‌ی وارونی از val_loss).

- monitor='val_loss': این تنظیم مشخص می‌کند که کدام معیار را باید برای مانیتورینگ استفاده کند. در اینجا از loss مربوط به داده‌های اعتبارسنجی استفاده می‌شود. زیرا بهتر از لاس معمولی است طبق تست های صورت گرفته!

- mode='min': این تنظیم مشخص می‌کند که مدل به ازای مقدار مینیمم monitor شده (در این حالت val_loss) ذخیره شود.

- verbose=۱: این تنظیم مشخص می‌کند که آیا اطلاعات بیشتری هنگام ذخیره‌سازی مدل نمایش داده شود یا خیر.

پس با استفاده از این callback، مدل شبکه عصبی پس از هر بار آموزش، مدل ذخیره و بهبودیافته‌ترین نسخه آن بر اساس مقدار val_loss ذخیره می‌شود.

٤-تاثیر توابع فعالسازی:

ReLU (Rectified Linear Activation): این تابع غیرخطی است و در لایه‌های مخفی شبکه عصبی به طور گسترده استفاده می‌شود. این تابع اعداد منفی را به صفر تبدیل می‌کند.

استفاده از تابع فعال‌سازی Relu (Rectified Linear Unit) به طور معمول در شبکه‌های عصبی عمیق (Deep Neural Networks) توصیه شده است، زیرا این تابع بهبود مهمی در آموزش و عملکرد مدل‌ها می‌آورد. از مزایای استفاده از Relu می‌توان به سرعت آموزش، جلوگیری از مشکل مواجهه با مشکل مرگ نورون (Vanishing Gradient Problem) و افزایش قدرت انتقال سیگنال‌های غیرخطی اشاره کرد. به همین دلیل استفاده از Relu به جای Sigmoid در شبکه‌های عصبی رایج تر است. در واقع سیگموید را میتوان با سافت مکس مقایسه کرد که در ادامه شکست ان را میبینیم زیرا برای مثال دو کلاسه خوب است!

Softmax: این تابع بیشتر برای مسائل طبقه‌بندی استفاده می‌شود. این تابع ورودی‌های خروجی را به احتمالات مقابله‌ای تبدیل می‌کند که مجموع آن‌ها برابر با ۱ است، بنابراین می‌توان احتمال تعلق هر ورودی به هر کلاس را مشخص کرد.

چرا Sigmoid استفاده نکردیم؟

استفاده از تابع فعال‌ساز softmax در لایه خروجی این شبکه عصبی MLP از تابع فعال‌ساز sigmoid بهتر است زیرا که مسئله دسته‌بندی چند دسته‌ای (multi-class classification) با مقادیر خروجی احتمالی برچسب‌ها را داریم. تابع softmax به خوبی بازه‌ی احتمالات را بیان میکند و این مسئله را مناسبتر می‌کند.

تابع sigmoid به عنوان تابع فعال‌ساز در مسائل دسته‌بندی دو دسته‌ای (binary classification) معمولاً استفاده می‌شود، زیرا اعداد را به بازه ۰ تا ۱ محدود می‌کند که متناسب با خروجی‌های تنها یک برچسب است.

برای مسائل دسته‌بندی چند دسته‌ای مانند MNIST که دارای ۱۰ کلاس است، استفاده از تابع softmax منطقی‌تر است زیرا توانایی مدل در پیش‌بینی احتمال هر یک از کلاس‌ها را فراهم می‌کند.

۵-نرخ یادگیری و الگوریتم های بهینه سازی:

در مدل‌های شبکه‌های عصبی، نرخ یادگیری (learning rate) میزانی است که مشخص می‌کند که چقدر وزن‌های شبکه در هر مرحله به سمت جواب بهینه تغییر کنند. این نرخ یادگیری می تواند بر اساس تجربه و تلاش های انجام شده توسط افراد و یا با استفاده از الگوریتم‌های بهینه‌سازی مشخص شود.

برای کد تایید نرخ یادگیری در مدل‌های شبکه‌های عصبی، معمولا از رویکردهای زیر استفاده می‌شود:

۱. Grid Search: با استفاده از روش Grid Search، می‌توان یک مجموعه از مقادیر نرخ یادگیری را تعیین کرده و سپس مدل را بر اساس هرکدام از این مقادیر آموزش داده و به دنبال بهترین عملکرد مدل با توجه به مقدار نرخ یادگیری باشیم.

۲. Random Search: در این روش، مقادیر نرخ یادگیری به صورت تصادفی انتخاب می‌شوند و مدل بر اساس این مقادیر آموزش داده می‌شود. این روش می‌تواند به صورت موثرتری مقدار بهینه را پیدا کند به واسطه جستجو در فضای مقادیر به صورت تصادفی.

۳. Optimization Algorithms: الگوریتم‌های بهینه‌سازی مانند Adam, RMSprop و SGD می‌توانند کمک کنند تا نرخ یادگیری بهینه برای مدل شبکه عصبی شما پیدا شود. که در اینجا ما از ادام برای بهبود نرخ یادگیر استفاده کردیم. که البته ادام به صورت دیفالت اگر نرخ یادگیری برایش تعیین نکنیم دیفالت ۰.۰۰۱ میگذارد یعنی :

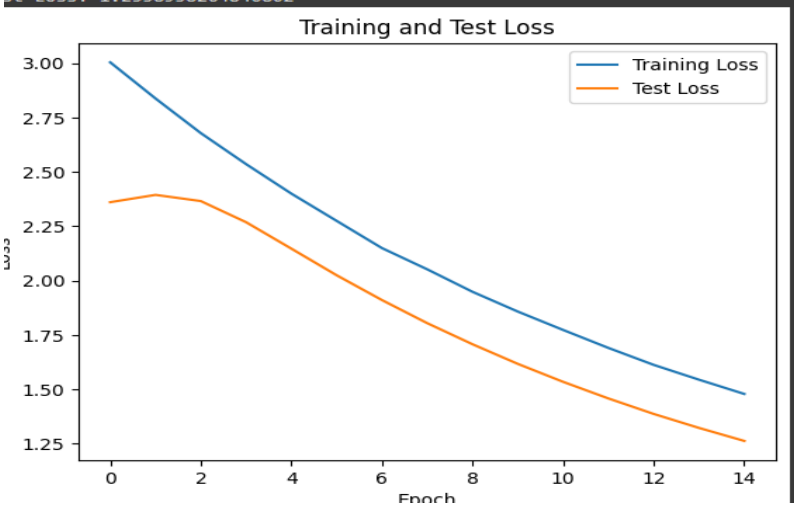
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

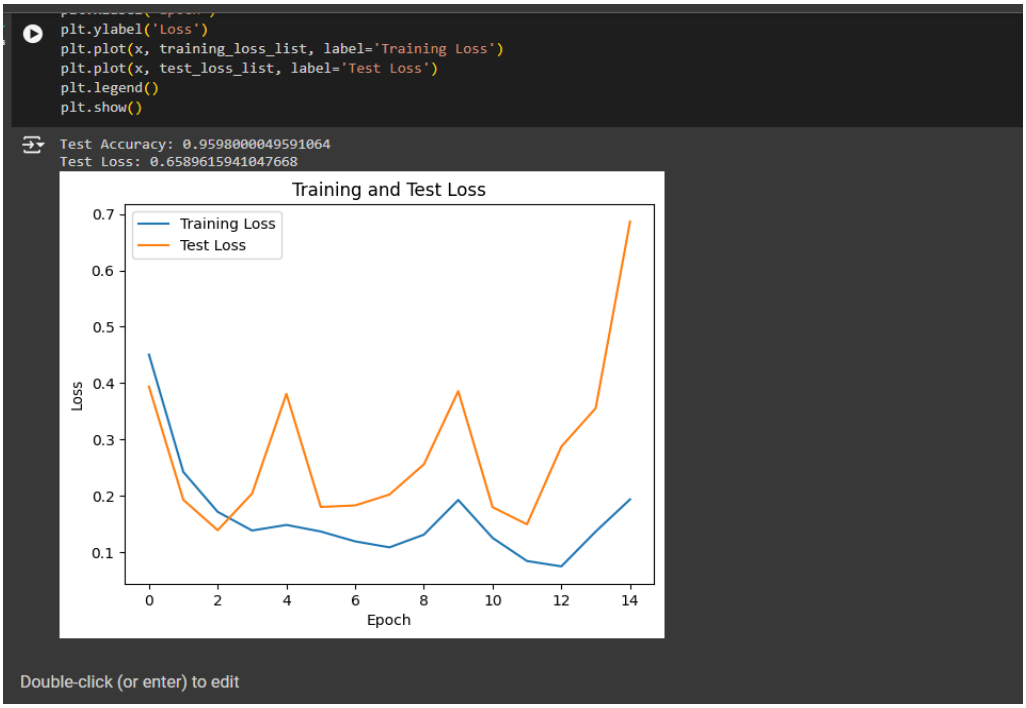
حال با تغییر این عدد و تست مقادیر مختلف مثلا ۰.۰۰۰۰۰۱=learning_rate باعث می شد دقت پایین تر رود درواقع در اکسترمم های محلی گیر کند و گام

برداشتن ان بسیار کوچک باشد. از طرفی هم بزرگ تر کردن ان سبب دور شدن ناگهانی از اکسترمم گلوبال بود و نمودار ها را از فیت دور تر می کرد:

```
st_loss_list = metrics['val_loss']

Plot the training and test loss.
= np.arange(0, num_epochs, 1)
t.title('Training and Test Loss')
t.xlabel('Epoch')
t.ylabel('Loss')
t.plot(x, training_loss_list, label='Training Loss')
t.plot(x, test_loss_list, label='Test Loss')
t.legend()
t.show()
```

```
st Accuracy: 0.6133999824523926
st Loss: 1.2558938264846802
```





پس در نهایت مقدار ۰.۰۰۱ را برگزیدم که به نسبت بر اساس تست و سرچ انتخاب مناسب تری به ازای تغییر اپوک ها و بچ نیز بود...!

بهینه‌سازی‌ها در اصل الگوریتم‌هایی هستند که هدف آن‌ها بهینه کردن پارامترهای مدل به نحوی است که تابع هدف کاهش یابد. بهینه‌سازها می‌توانند با تنظیم نرخ یادگیری، انجام بهینه‌سازی محلی یا جلوگیری از گیر کردن در نقاط مینیمم موجود، اهمیت بیشتری به انتقال سریع‌تر یا پایدار تر به مینیمم بدهند.

در مورد انتخاب الگوریتم بهینه‌سازی، تجربه، آزمون و خطا، و خصوصیات مدل و مساله می‌تواند به تصمیم نهایی کمک کند. بهینه‌سازی مناسب می‌تواند منجر به آموزش بهتر و سریع‌تر مدل شود، اما همیشه لازم است نکاتی همچون افزایش یا کاهش نرخ یادگیری را نیز در نظر گرفت.

برای آموزش مدل MLP بر روی دیتاست MNIST، می‌توان از بهینه‌سازی‌های مختلفی مانند RMSprop یا SGD نیز استفاده کرد. انتخاب بهینه‌سازی مناسب بستگی به ویژگی‌های خاص مدل، مساله و دیتاست دارد.

- RMSPROP یک روش بهینه‌سازی است که از شبکه عصبی برای آموزش با داده‌های بزرگ استفاده می‌شود. این الگوریتم از نسبت تغییرات گرادیان را برای هر وزن استفاده می‌کند تا مقدار learning rate را تطبیق دهد. با استفاده از این متد، روشی موثر برای جلوگیری از شلیک زودرس هنگام یادگیری عمیق است. ولی طبق نتایج و مباحث یاد گرفته ادام از ان بهتر است زیرا یک روش ترکیبی از این و یک الگوریتم دیگر است پس قطعا این مورد از لیست انتخاب ها حذف می شود.

- SGD_M یک نوع از روش Stochastic Gradient Descent (SGD) است که از مفهوم momentum برای سرعت بخشیدن به فرایند یادگیری استفاده می‌کند. این الگوریتم از مفهوم گذشته گرادیان‌ها برای بهبود سرعت یادگیری استفاده می‌کند تا از مشکلات سرعت کوهیدن گرادیان در بهینه‌سازی SGD معمولی کاسته شود. این روش می‌تواند بهبود قابل توجهی در سرعت و کیفیت یادگیری شبکه‌های عصبی داشته باشد. پس این روش نیز از SGD بهتر است اما باز هم از ادام ضعیف تر زیرا طبق تئوریات ادام ترکیبی از انهاست و بهتر عمل می کند. حتی با جایگذاری انها به جای ادام از دقت بالای ۹۰ درصدی کاسته شد!

- Adam یک الگوریتم بهینه‌سازی است که ترکیبی از روش‌های RMSprop و Momentum است. Adam با استفاده از میانگین ریاضی و تجهیز شده‌ی گرادیان‌ها بهبودی بهینه‌سازی بخصوص در مسائل با مقیاس حداقلی دارد.

همانطور که در درس ماشین لرنینگ خواندیم این الگوریتم به طور کلی می‌تواند به صورت موثری در مقایسه با الگوریتم‌های دیگر عمل کند و به سرعت و کارآیی مدل کمک کند.

نوع توابع هزینه و معیارها (Metrics) بستگی به نوع مسأله یادگیری ماشین دارد.

"لاس (loss)" میزان خطای تخمینی مدل در هر مرحله از آموزش است که به منظور بهبود عملکرد مدل کاهش داده می‌شود.

برخی از توابع هزینه معروف شامل:

۱. Binary Crossentropy (دسته‌بندی دودویی)

- استفاده معمولی برای مسائل تصمیم‌گیری دودویی است.

۲. Categorical Crossentropy (دسته‌بندی چند دسته‌ای)

- معمولا برای آموزش مدل‌هایی که باید داده‌ها را به یکی از چند دسته تقسیم کنند، مورد استفاده قرار می‌گیرد.

۳. Mean Squared Error (خطا میانگین مربعات)

- معمولا برای مسائل رگرسیون استفاده می‌شود.

۴. Kullback-Leibler Divergence (انحراف کولباک-لایبلر)

- برای مدل‌های توزیع احتمالاتی و یادگیری نظارت شده به‌خصوص مسائل تولید محتوا مانند مولدهای مقابله‌ای (GANs) استفاده می‌شود.

"متریک (metrics)" به معنای معیارهایی است که برای ارزیابی عملکرد مدل در هنگام آموزش یا آزمون استفاده می‌شود، مانند دقت، دقت خاصیتی و ...

معیارها نیز برای ارزیابی مدل استفاده می‌شوند. علاوه بر دقت، معیارهای دیگری نیز وجود دارند که می‌توان در مورد عملکرد مدل استفاده کرد مانند:

- فراخوانی (recall)

- دقت (precision)

- اف اسکور (F1-score)

- ماتریس درهم‌ریختگی (confusion matrix) و...

با توجه به نوع مسأله و نوع داده‌ها، انتخاب صحیح توابع هزینه و معیارهای مناسب بسیار حیاتی است. اما انتخاب categorical_crossentropy و accuracy در این مسئله ی خاص مناسب ترین حالت می تواند باشد.

- loss=categorical_crossentropy: این بخش مشخص می‌کند که برای اندازه‌گیری خطا یا هزینه در حین آموزش از تابع هزینه‌ی "categorical_crossentropy" استفاده شود. این تابع مخصوص کاربردی است که برای مسائل دسته‌بندی چند دسته‌ای مناسب است.

البته که یک نکته دیگر هم داریم:

اگر داده‌ها به صورت integer labels هستند، انتخاب sparse_categorical_crossentropy مناسب است. اما اگر داده‌ها تبدیل به one-hot encode شده‌اند، انتخاب categorical_crossentropy صحیح است. که در اینجا چون هات انکود کردیم پس کتگوریکال بهتر است!

- metrics=['accuracy']: این بخش به مدل مشخص می‌کند که در هر مرحله از آموزش، دقت (accuracy) را به عنوان معیار برای ارزیابی عملکرد مدل استفاده

کند. که دقت نشان دهنده درصد داده‌هایی است که به درستی تشخیص داده شده‌اند.

٦-تعیین پارامتر و نرخ ها:

```
# Define the parameters.
num_epochs = 30
batch_size = 256

# Train the model.
history = model.fit(X_train,
                    y_train,
                    epochs=num_epochs,
                    batch_size=batch_size,
                    validation_data=(X_test, y_test))
```

۱. num_epochs: این پارامتر تعداد دوره‌های آموزش را مشخص می‌کند، به این معنی که داده‌ها به مدل به مدت ۳۰ بار آموزش داده می‌شوند. این مقدار بر اساس تست چندین عدد و تئوریات کتاب اصلی بر اساس پیچیدگی مسئله و حجم داده‌ها تنظیم کردم و نه اجرا انقدر کند و نفس گیر شد تا فیت شود و نه انقدر کم و سریع بود که دقت پایین داشته باشد یعنی به یک نتیجه ی مطلوب در احتمالا مناسب ترین عدد رسیدیم. عداد نزدیک ۳۰ نیز یا دقت را پایین می اورد یا چندان سبب بهبود ان نبود و فقط سرعت را پایین می اورد.

۲. batch_size = ۲۵۶: این پارامتر تعداد داده‌هایی که به صورت همزمان به مدل وارد می‌شود را مشخص می‌کند. استفاده از دسته‌های کوچک (batch) از داده‌ها بهینه‌سازی فرآیند آموزش را کمک می‌کند، زمانی که داده‌های زیادی داریم. به دلیل تعادل بین سرعت آموزش و حافظه ۲۵۶ استفاده شده است. انتخاب اندازه batch معمولاً یک ترید اف بین عملکرد و سرعت است. انتخاب اندازه batch بزرگتر از یک، می‌تواند کمک کند تا برای همه داده‌ها یک بار گرادیان‌ها

محاسبه شود ولی از حافظه بیشتری استفاده کند. از سوی دیگر، انتخاب اندازه batch کوچکتر می‌تواند منجر به یک فرآیند آموزش ناپایدارتر شود ولی از حافظه کمتری استفاده کند. در اینجا با تنظیم اندازه batch به ۲۵۶، تلاش برای تعادل بین استفاده از حافظه و سرعت آموزش می‌شود. برای مجموعه داده MNIST که نسبتاً کوچک است، استفاده از یک اندازه batch بزرگتر مانند ۲۵۶ ممکن است به صورت عملی باشد و به سرعت آموزش کمک کند بدون اینکه به نحو چشمگیری از مقدار حافظه استفاده شود.

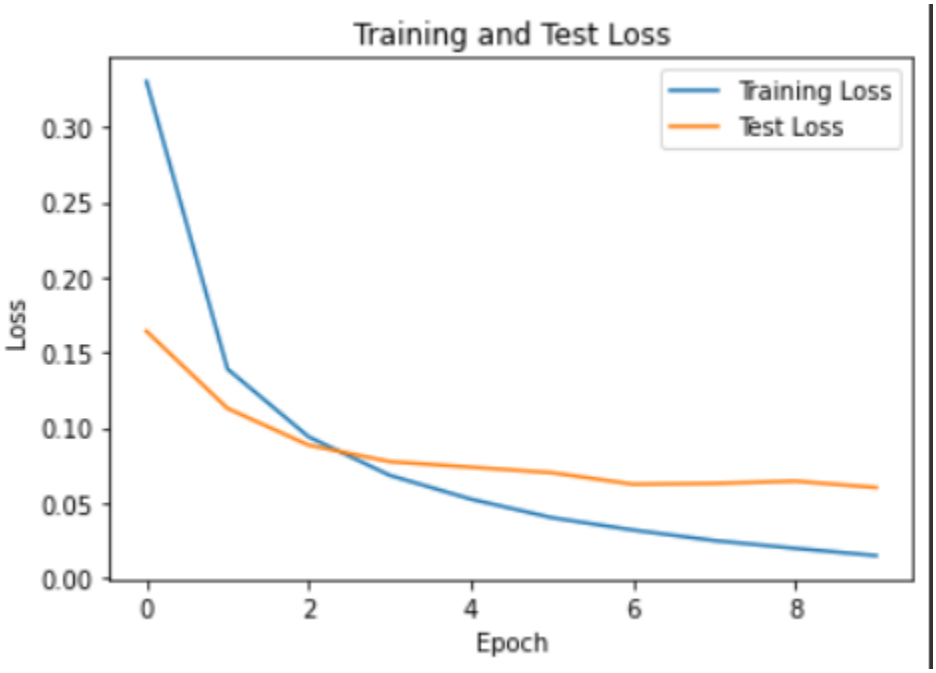
۷-رسم نمودار های مربوطه:

- برای تفسیر عملکرد کلی مدل از پلات های پایتون کمک گرفتم

```
# Save the loss values.
training_loss_plot = metrics['loss']

test_loss_plot = metrics['val_loss']

# Plot the training and test loss.
x = np.arange(0, numbers_of_epochs, 1)
plt.title('Training and Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(x, training_loss_plot, label='Training Loss')
plt.plot(x, test_loss_plot, label='Test Loss')
plt.legend()
plt.show()
```

loss (که مربوط به loss در مرحله‌ی آموزش باشد) معمولا به میانگین خطای محاسبه شده بر روی داده‌های آموزش اشاره دارد، در حالی که val_loss مربوط به خطای محاسبه شده بر روی داده‌های اعتبارسنجی یا ارزیابی (validation set) است. پس درواقع علاوه بر ترین کردن یک تست روی داده های ترین نیز رخ داده که همان ولیدیت کردن می باشد.

اختلاف بین این دو معمولا نشان دهنده‌ی عملکرد مدل در داده‌های دیده نشده‌ی validation set نسبت به داده‌های آموزش است. اگر val_loss بیشتر از loss باشد، مدل ممکن است دچار overfitting (برازش بیش از حد) شده باشد، به این معنا که در حالتی شده که به داده‌های آموزش بسیار خوب عمل کند اما در داده‌های جدید (مانند داده‌های اعتبارسنجی) عملکرد بهتری نداشته باشد.

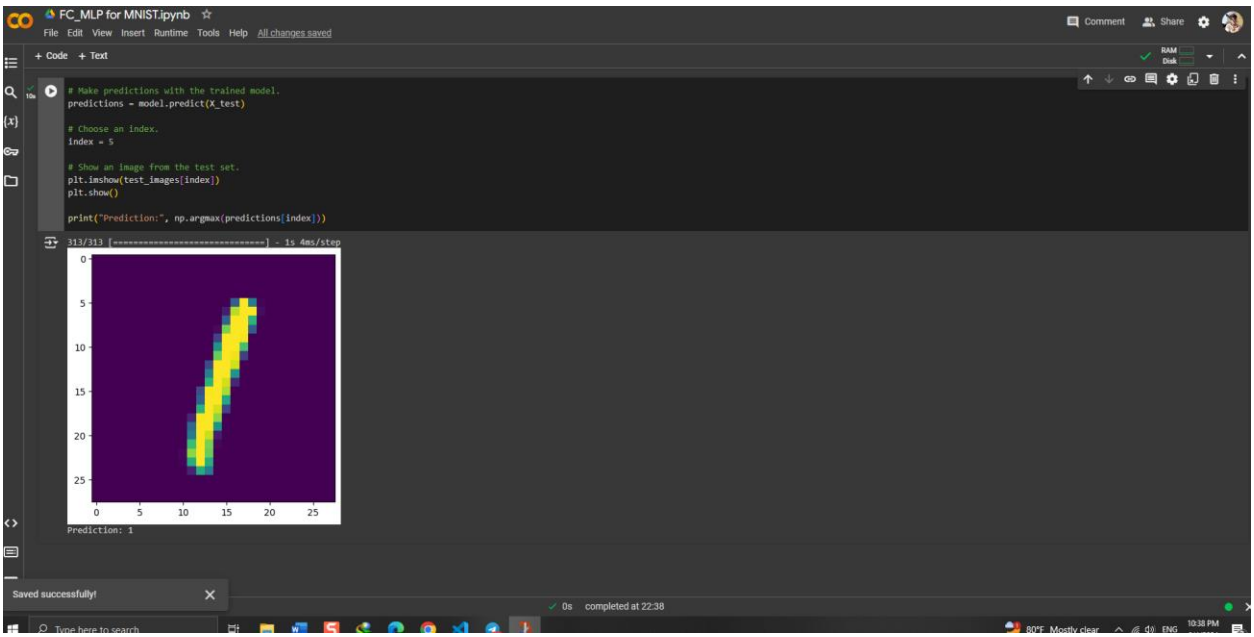


همین روال را علاوه بر خطا برای دقت نیز محاسبه و رسم می کنیم:

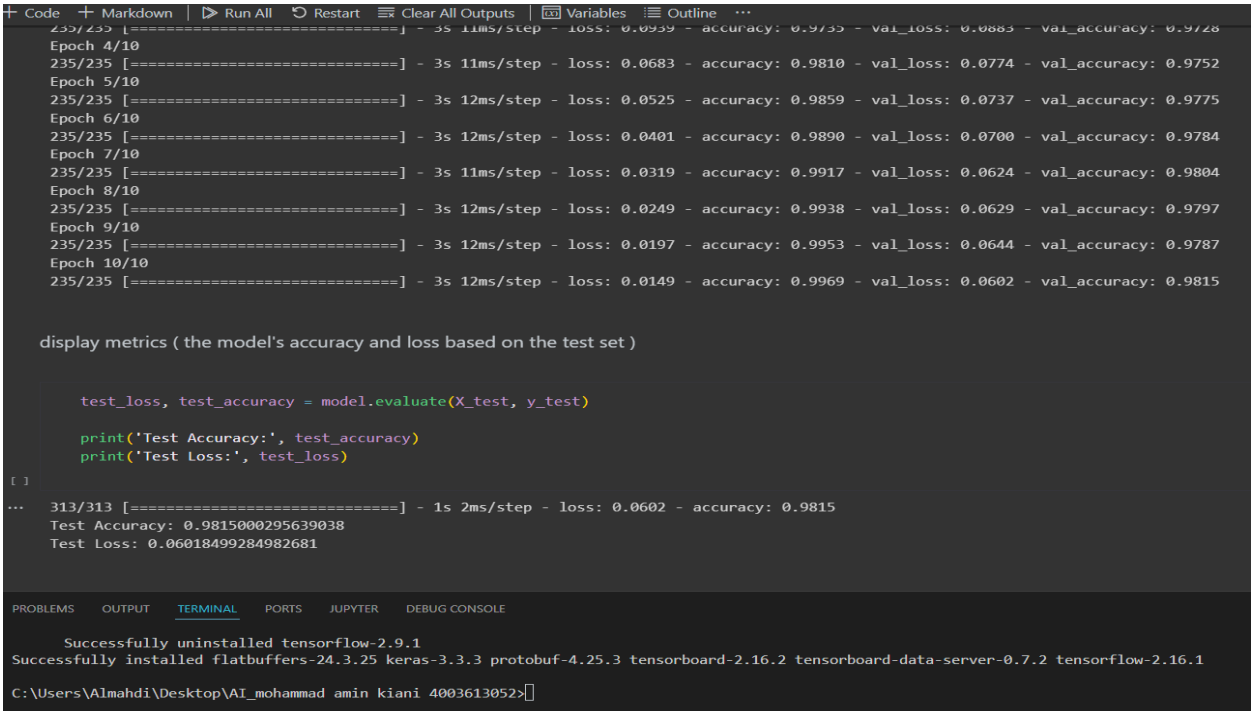
```
train_accuracy_list = metrics['accuracy']

test_accuracy_list = metrics['val_accuracy']

plt.title('Training and Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.plot(x, train_accuracy_list, label='Training Accuracy')
plt.plot(x, test_accuracy_list, label='Test Accuracy')
plt.legend()
plt.show()
```

همانطور که از نمودار ها پیداست یعنی میزا اختلاف ارتفاع تست و ترین به میزان مناسب و معقولی است یعنی نه اورفیت رخ داده و نه اندرفیت و مدل به درستی فیت شده است!

۸-فیت شدن و پیشگویی هر داده با آن:



۹-خروجی نهایی:



٩-مراجع

[1] https://github.com
[2] https://stackoverflow.com/questions
[3] https://www.wikipedia.org/
[4] https://colab.research.google.com/
[5] https://www.tensorflow.org/guide/
[6] https://pandas.pydata.org/
[7] https://keras.io/
[8] https://www.projectpro.io/article/exploring-mnist-dataset-using-pytorch-to-train-an-mlp/408
[9] https://stats.stackexchange.com/questions/376312/mnist-digit-recognition-what-is-the-best-we-can-get-with-a-fully-connected-nn-o