|  |  |
| --- | --- |
| پیاده سازی MLP | پیاده سازی شبکه چندلایه ای  در این گزارش سعی کردیم یک شبکه عصبی چندلایه به کمک کتابخانه های پایتون پیاده سازی کنیم و با استفاده از متغیرهای گوناگون برای آموزش این شبکه بهینگی و کارایی آن را مورز آزمایش قرار دادیم.  سیدمحمدجواد موسوی  استاد بهزاد پورمحمود |

در ابتدا به کمک کتابخانه Tensorflow دیتاست fashion\_mnist را در (x\_train, y\_train) و (x\_test, y\_test) لود می‌کنیم که شامل عکس هایی برای آموزش و آزمون مدل ماست.

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.fashion\_mnist.load\_data()

در ادامه به Preprocess داده ها می‌پردازیم:

* ابتدا با تقسیم داده ورودی ها یعنی x\_train و x\_test بر 255 مقدار پیکسل ها را به رقمی بین 0 و 1 کاهش می‌دهیم تا داده ها نرمالایز شوند.
* در ادامه label ها را به کمک متد to\_categorical به شکل one-hot encoding در می‌آوریم. (گفتنی است که این کار کمک می‌کند تا خروجی ها از حالت رشته ای خارج شوند و یک آرایه تبدیل

شوند تا ترتیب آن ها از بین برود)

x\_train = x\_train / 255.0  
x\_test = x\_test / 255.0  
y\_train = keras.utils.to\_categorical(y\_train)  
y\_test = keras.utils.to\_categorical(y\_test)

حال مدل را طراحی می‌کنیم: به کمک تابع sequential از کلاس keras شبکه عصبی دو لایه میسازیم:

* یک لایه flatten که یک آرایه دوبعدی را به یک وکتور تک بعدی تبدیل می‌کند.
* یک لایه dense با 128 نورون و تابع فعال سازی ReLU
* یک لایه dense با 10 نورون و تابع فعال سازی softmax

گفتنی است که در ازمایش های مختلف این توابع و تعداد نورون ها عوض میشوند و نتایج گزارش می‌شود.

model = keras.Sequential([  
 keras.layers.Flatten(input\_shape=(28, 28)),  
 keras.layers.Dense(128, activation='relu'),  
 keras.layers.Dense(10, activation='softmax')  
])

سپس مدل را برای انجام مرحله آموزش کامپایل می‌کنیم:

* ابتدا بهینه ساز را تعیین می‌کنیم که adam است.
* سپس loss function را categorical\_crossentropy تعیین می‌کنیم.
* سپس تابع سنجش دقت را معادل accuaracy قرار می‌دهیم.

گفتنی است که در آزمایش های مختلف توابع Loss و بهینه ساز تغییر خواهند کرد و نتایج گزارش می‌شود.

model.compile(optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])

در مواقعی که می‌خواهیم از مومنتوم استفاده کنیم این تکه کد را اضافه می‌کنیم و به عنوان بهینه ساز به کامپیال می‌دهیم:

opt = keras.optimizers.Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-07, amsgrad=False, momentum=0.9)

حال مدل را با 10 epoch و دیتا های آموزش، آموزش می‌دهیم و برای نمایش نتایج در ادامه مقدار بازگشتی تابع را در history ذخیره می‌کنیم.

گفتنی است که تعداد epoch ها در در آزمایش های مختلف تغییر داده شده و نتایج گزارش می‌شوند.

history = model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))

در ادامه نتایج حاصل را به کمک کتابخانه matplotlib نمایش می‌دهیم

* ابتدا دقت را بر حسب پیشروی epoch ها نمایش می‌دهیم
* سپس Loss را بر حساب پیشروی epoch ها نمایش می‌دهیم