

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه اول درس مبانی هوش محاسباتی

پیاده‌سازی شبکه عصبی تمام متصل دیتاست MNIST

مهرآذین مرزوق – ۴۰۰۳۶۱۳۰۵۵

فهرست

[انتخاب تعداد لایه‌ها 3](#_Toc169279329)

[انتخاب تعداد نورون‌های هر لایه 5](#_Toc169279330)

[انتخاب الگوریتم بهینه‌سازی 7](#_Toc169279331)

[انتخاب نرخ یادگیری 10](#_Toc169279332)

[تاثیر overfitting و underfitting 12](#_Toc169279333)

[شرایط توقف 13](#_Toc169279334)

[تاثیر activation function 15](#_Toc169279335)

[تاثیر dropout 16](#_Toc169279336)

[تاثیر batch normalization 18](#_Toc169279337)

# انتخاب تعداد لایه‌ها

تعداد مناسب لایه برای یک پرسپترون چند لایه به چندین فاکتور وابسته است:

پیچیدگی دیتاست MNIST: این دیتاست تقریبا دیتاست ساده‌ای است. اما پرسپترون چندلایه می‌تواند الگوهای پیچیده‌تری را یاد بگیرد. بنابراین اگر تعداد لایه‌ها اندکی بیشتر شود، به سرعت overfitting رخ می‌دهد.

معمولا برای دیتاست MNIST از ۲ یا ۳ لایه استفاده می‌شود؛ اما باید توجه داشت که در صورتی که از ۳ لایه استفاده کنیم، باید بیشتر مراقب بیش‌برازش باشیم.

دو لایه: لایه اول ویژگی‌ها را استخراج می‌کند و لایه دوم، ویژگی‌ها را برای classification ترکیب می‌کند.

# Add two fully-connected layers to the network

model.add(tf.keras.layers.Dense(512, activation='relu', input\_shape=(28 \* 28,)))

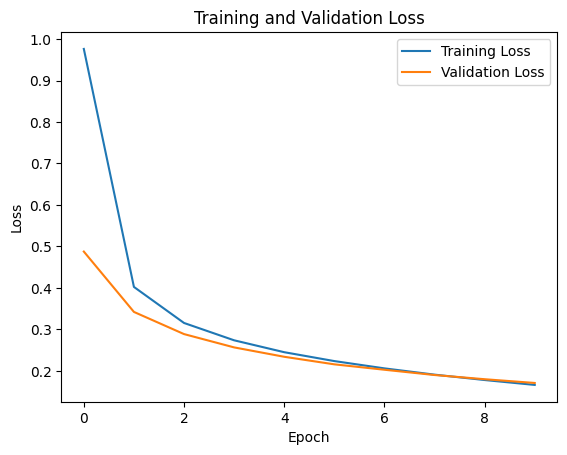
model.add(tf.keras.layers.Dense(10, activation='softmax'))

Test Accuracy: 0.9534000158309937

Test Loss: 0.16711154580116272

Validation Accuracy: 0.9519000053405762

Validation Loss: 0.17070327699184418

A graph of a graph

Description automatically generated with medium confidence

سه لایه:

# Add three fully-connected layers to the network.

model.add(tf.keras.layers.Flatten(input\_shape=(28 \* 28,)))

model.add(tf.keras.layers.Dense(32, activation='relu'))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

Test Accuracy: 0.9079999923706055

Test Loss: 0.3306991159915924

Validation Accuracy: 0.9064000248908997

Validation Loss: 0.340914785861969

A graph of a graph

Description automatically generated

در اینجا مشاهده می‌شود که با ۳ لایه، دقت پایین می‌آید. به علت این‌که این دیتاست، دیتاست ساده‌ای است و استفاده از ۳ لایه باعث افزایش پیچیدگی هم در پیاده‌سازی و هم در اجرا می‌شود، تصمیم بر این می‌شود که از ۲ لایه استفاده کنیم.

# انتخاب تعداد نورون‌های هر لایه

در رابطه با دیتاست MNIST در صورتی که از ۲ لایه استفاده شود، معمولا از ۵۱۲ نورون در لایه اول استفاده می‌شود. ما این عدد را در نظر میگیرم و با بالا و پایین کردن این عدد، به بررسی نتایج می‌پردازیم.  
معیار ما برای مناسب بودن تعداد نورون‌های لایه اول، validation accuracy و train accuracy را مقایسه می‌کنیم. اعداد بالای ۰.۹۵ که در این دو به هم نزدیک باشند، به معنی این است که مدل به خوبی داده را آموخته است و از بیش برازش جلوگیری می‌کند. همچنین بررسی می‌کنیم که test accuracy در حال افزایش و test loss درحال کاهش باشد.

۵۱۲:

Test Accuracy: 0.953499972820282

Test Loss: 0.16269086301326752

Validation Accuracy: 0.9532999992370605

Validation Loss: 0.16897334158420563

Train Accuracy: 0.957099974155426

Train Loss: 0.1573188155889511

۲۵۶:

Test Accuracy: 0.9430999755859375

Test Loss: 0.2028328776359558

Validation Accuracy: 0.9447000026702881

Validation Loss: 0.20519550144672394

Train Accuracy: 0.9460399746894836

Train Loss: 0.1990952342748642

در اینجا test accuracy نسبت به قبل افزایش پیدا کرده، validation accuracy و test accuracy از ۰.۹۵ کمتر و نسبت به قبل کاهش پیدا کرده‌اند.

این نشان می‌دهد که تعداد نورون کمتر از ۵۱۲ باعث underfitting در مدل می‌شود.

۱۰۲۴:

Test Accuracy: 0.9624999761581421

Test Loss: 0.12894481420516968

Validation Accuracy: 0.9629999995231628

Validation Loss: 0.13496488332748413

Train Accuracy: 0.9688799977302551

Train Loss: 0.11788497120141983

۲۰۴۸:

Test Accuracy: 0.9677000045776367

Test Loss: 0.10813374072313309

Validation Accuracy: 0.9692999720573425

Validation Loss: 0.11196314543485641

Train Accuracy: 0.9769799709320068

Train Loss: 0.08993370831012726

این تعداد نورون پیچیدگی زمانی زیادی دارد.

۴۰۹۶:

Test Accuracy: 0.9746999740600586

Test Loss: 0.08622105419635773

Validation Accuracy: 0.973800003528595

Validation Loss: 0.09152911603450775

Train Accuracy: 0.9848999977111816

Train Loss: 0.0626753717660904

این تعداد نورون با اینکه دقت را بالا می‌برد، اما پیچیدگی زیادی دارد و اجرای آن از نظر زمانی و سخت‌افزاری به‌صرفه نیست.

در نهایت تعداد ۱۰۲۴ نورون برای لایه اول انتخاب می‌شود.

در لایه دوم، به علت اینکه ۱۰ تا کلاس خواهیم داشت، از ۱۰ نورون استفاده می‌کنیم.

# انتخاب الگوریتم بهینه‌سازی

Adam: این الگوریتم یک الگوریتم کاهش گرادیان تصادفی است؛ از نظر محاسباتی بهینه است، به حافظه کمی نیاز دارد، نسبت به مقیاس مجدد مورب گرادیان‌ها ثابت است و برای مسائلی که تعداد زیادی دیتا و پارامتر وجود دارد مناسب است. کد زیر، کد پیشفرض کتابخانه keras برای الگوریتم adam می‌باشد. برای تست کردن، از مقادیر پیشفرض استفاده می‌کنیم.

) keras.optimizers.Adam(

learning\_rate=0.001,

beta\_1=0.9,

beta\_2=0.999,

epsilon=1e-07,

amsgrad=False,

weight\_decay=None,

clipnorm=None,

clipvalue=None,

global\_clipnorm=None,

use\_ema=False,

ema\_momentum=0.99,

ema\_overwrite\_frequency=None,

loss\_scale\_factor=None,

gradient\_accumulation\_steps=None,

name="adam",

\*\*kwargs

)

# Compile the model.

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

Test Accuracy: 0.9803000092506409

Test Loss: 0.0634884163737297

Validation Accuracy: 0.9800999760627747

Validation Loss: 0.06448717415332794

Train Accuracy: 0.9991199970245361

Train Loss: 0.008130251429975033

RMSProp: این الگوریتم از نظر محاسباتی، به ویژه زمانی که با مجموعه داده های بزرگ سروکار داریم، بهینه است. با استفاده از یک مثال واحد یا یک دسته کوچک، هزینه محاسباتی در هر تکرار در مقایسه با روش‌های سنتی گرادیان نزولی که نیاز به پردازش کل مجموعه داده دارند، به میزان قابل توجهی کاهش می‌یابد. کد زیر، کد پیشفرض کتابخانه keras برای الگوریتم RMSProp می‌باشد. برای تست کردن، از مقادیر پیشفرض استفاده می‌کنیم.

keras.optimizers.RMSprop(

learning\_rate=0.001,

rho=0.9,

momentum=0.0,

epsilon=1e-07,

centered=False,

weight\_decay=None,

clipnorm=None,

clipvalue=None,

global\_clipnorm=None,

use\_ema=False,

ema\_momentum=0.99,

ema\_overwrite\_frequency=None,

loss\_scale\_factor=None,

gradient\_accumulation\_steps=None,

name="rmsprop",

\*\*kwargs

# Compile the model.

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

Test Accuracy: 0.9819999933242798

Test Loss: 0.06130603700876236

Validation Accuracy: 0.9810000061988831

Validation Loss: 0.06498846411705017

Train Accuracy: 0.9985600113868713

Train Loss: 0.008354941383004189

مقایسه این دو مدل:

adam

* مزایا:
  + اغلب سریعتر از RMSprop همگرا می شود
  + دارای نرخ یادگیری تطبیقی با مومنتوم است که با مسائل گرادیان های در حال ناپدید شدن/منفجر شدن مقابله می کند
  + در بسیاری از چارچوب های یادگیری عمیق به عنوان بهینه ساز پیش فرض در نظر گرفته می شود
* معایب:
  + ممکن است نسبت به RMSprop به محاسبات کمی بیشتر نیاز داشته باشد

RMSprop

* مزایا:
  + گرادیان های پراکنده را که در شبکه های عصبی رایج هستند به طور موثر مدیریت می کند
  + به طور کلی از نظر محاسباتی کم هزینه تر از Adam است
* معایب:
  + ممکن است در برخی موارد کندتر از Adam همگرا شود
  + از تصحیح سوگیری Adam که می تواند در مراحل بعدی آموزش مفید باشد، بی بهره است

با توجه به سادگی مجموعه داده MNIST و یک MLP دو لایه، هر دو بهینه ساز ممکن است به نتایج خوبی دست یابند. با این حال، adam اغلب به دلیل موارد زیر به عنوان پیش فرض توصیه می شود:

* همگرایی سریعتر (احتمالی)
* یادگیری تطبیقی با مومنتوم
* اثربخشی به عنوان یک بهینه ساز چند منظوره

# انتخاب نرخ یادگیری

نرخ یادگیری میزان قدم‌های مدل را که برای به‌دست آوردن وزن سیناپس‌ها استفاده می‌شود، مشخص می‌کند.

نرخ یادگیری به‌طور پیش‌فرض در الگوریتم بهینه‌ساز مورد نظر انتخاب شده است. نرخ یادگیری پیش‌فرض در الگوریتم adam همانطور که در بخش قبل مشاهده شد، برابر با ۰.۰۰۱ است.

نرخ یادگیری کوچک (۰.۰۰۰۱) پایداری بیشتری دارد و همچنین شانس بیشتری برای پیدا کردن global minimum دارد. اما یادگیری کندتر است ( و گاهی ممکن است که در shallow minima گیر کند).

نرخ یادگیری بزرگ یادگیری سریعتری دارد اما ناپایدار است و ممکن است در local minimum گیر کند.

در این پروژه ابتدا از ۰.۰۰۰۱ شروع می‌کنیم و نتایج مربوط به loss را بررسی می‌کنیم. اگر validation loss افزایش پیدا کند، نرخ یادگیری احتمالا بزرگ است و اگر ثابت بماند، بسیار کوچک است.

۰.۰۰۰۱:

Test Accuracy: 0.9623000025749207

Test Loss: 0.13262595236301422

Validation Accuracy: 0.9620000123977661

Validation Loss: 0.1367652416229248

Train Accuracy: 0.968280017375946

Train Loss: 0.12076471000909805

۰.۰۰۱:

Test Accuracy: 0.9825000166893005

Test Loss: 0.061019714921712875

Validation Accuracy: 0.9812999963760376

Validation Loss: 0.05988801270723343

Train Accuracy: 0.9994000196456909

Train Loss: 0.006334839388728142

۰.۰۱:

Test Accuracy: 0.9707000255584717

Test Loss: 0.16833384335041046

Validation Accuracy: 0.9695000052452087

Validation Loss: 0.17394115030765533

Train Accuracy: 0.9880399703979492

Train Loss: 0.04146776348352432

طبق توضیحات قبل، بهترین عدد ۰.۰۰۱ است که پیش‌فرض adam است.

# # Compile the model.

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.001)

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

# تاثیر overfitting و underfitting

Overfitting

زمانی رخ می دهد که مدل شما جزئیات و نویز خاصی را که در داده های آموزشی وجود دارد به خاطر می آورد. به علاوه اینکه الگوهای زمینه‌ای را که به نمونه های دیده نشده تعمیم می یابد، یاد بگیرد.

* علائم
  + دقت آموزش بالا اما دقت بسیار پایین در داده‌های validation یا test است.
* چگونه با MNIST اتفاق می افتد:
  + استفاده از یک مدل پیچیده با پارامترهای بسیار زیاد برای اندازه مجموعه داده MNIST می تواند منجر به overfitting شود.
  + فقدان تکنیک های افزایش داده (به عنوان مثال، چرخش های تصادفی، تزریق نویز) می تواند توانایی مدل را برای تعمیم به تغییراتی که در مجموعه آموزشی وجود ندارد محدود کند.

Underfitting

زمانی رخ می دهد که مدل شما بسیار ساده باشد و از ظرفیت یادگیری پیچیدگی های موجود در داده ها برخوردار نباشد.

* علائم
  + دقت پایین در مجموعه های train و validation/test
  + مدل ممکن است حتی در تمایز بین اشکال اولیه رقم نیز با مشکل مواجه شود.
* چگونه با MNIST اتفاق می افتد:
  + استفاده از مدلی با لایه های بسیار کم یا نورون های بسیار کم می تواند توانایی آن را برای گرفتن ویژگی های لازم برای طبقه بندی دقیق رقم محدود کند.
  + آموزش برای تعداد غیربهینه از epochs ممکن است مانع از یادگیری کامل الگوهای موجود در داده ها توسط مدل شود.

استراتژی هایی برای اجتناب از overfittingو underfitting

* مجموعه validation : یک مجموعه اعتبارسنجی نگه داشته شده به شما امکان می دهد عملکرد مدل را روی داده های دیده نشده در طول آموزش کنترل کنید. این به شناسایی زودهنگام overfitting کمک می کند.
* توقف زودهنگام: هنگامی که افت اعتبارسنجی ثابت شد، آموزش را متوقف کنید تا از به خاطر سپردن نویز در داده های آموزشی توسط مدل جلوگیری شود.
* انتخاب مدل: با معماری های مدل مختلف (تعداد لایه ها، نورون ها) و پارامترهای آموزشی آزمایش کنید.

# شرایط توقف

در چنین پروژه‌ای، شرایط توقف به ۴ نوع می‌توانند باشند.

نوع اول: توقف زودهنگام

در این روش، با بررسی validation loss و validation accuracy ، اگر تغییرات این دو از epoch ای به بعد ناچیز یا منفی بود، یادگیری را متوقف می‌کنیم که وارد فاز overfitting نشویم

نوع دوم: رسیدن به یک validation accuracy مشخص

در این روش، اگر زودتر از تعداد epoch مورد نظر به accuracy مورد انتظار رسیدیم، یادگیری را متوقف می‌کنیم که وارد فاز overfittng نشویم.

نوع سوم: بیشترین epoch

در این روش، عددی را برای تعداد epoch ارائه می‌دهیم که نه underfitting و نه overfitting داشته باشیم.

نوع چهارم: کاهش نرخ یادگیری

در این روش، ابتدا مدل گام‌های بزرگی برای یادگیری برمی‌دارد و با گذشت زمان، این گام‌ها را کوچک‌تر می‌کند زیرا که احتمالا در آن زمان، به global minima نزدیک‌تر شده است و باید با احتیاط بیشتری حرکت کند. این کار باعث می‌شود که نسبت به نرخ یادگیری ثابت کم، پیچیدگی زمانی کمتر و نسبت به نرخ یادگیری زیاد، دقت بیشتری داشته باشیم.

در این پروژه، از روش بیشترین epoch استفاده می‌کنیم. این روش، درون خود روش اول را نیز دارد. روش چهارم نیز برای این دیتاست، اضافه‌کاری محسوب می‌شود.

۵:

Test Accuracy: 0.9769999980926514

Test Loss: 0.07319962978363037

Validation Accuracy: 0.9764000177383423

Validation Loss: 0.07930747419595718

Train Accuracy: 0.9902799725532532

Train Loss: 0.03488082066178322

۸:

Test Accuracy: 0.9799000024795532

Test Loss: 0.06312963366508484

Validation Accuracy: 0.9810000061988831

Validation Loss: 0.06327541172504425

Train Accuracy: 0.9984800219535828

Train Loss: 0.011503975838422775

۱۰:

Test Accuracy: 0.9810000061988831

Test Loss: 0.06145879998803139

Validation Accuracy: 0.9818000197410583

Validation Loss: 0.06576941162347794

Train Accuracy: 0.9991000294685364

Train Loss: 0.0072038243524730206

۱۵:

Test Accuracy: 0.9814000129699707

Test Loss: 0.06390372663736343

Validation Accuracy: 0.9836000204086304

Validation Loss: 0.0610683411359787

Train Accuracy: 0.9999600052833557

Train Loss: 0.0017555037047713995

در اینجا بنظر می‌رسد که تعداد ۱۰ epoch مناسب باشد. ۱۵ epoch زمان زیادی را مصرف می‌کند و کمتر از ۱۰ هم دقت کمی دارد.

# تاثیر activation function

توابع فعال‌سازی موجود در keras عبارتند از : relu ،‌ softmax و sigmoid

Relu معمولا برای لایه‌های پنهان استفاده می‌شود چرا که به خوبی روابط غیر خطی را متوجه می‌شود. اگر خروجی مثبت باشد، همان را نمایش می‌دهد وگرنه، صفر نمایش می‌دهد.

Leaky relu: اجازه می‌دهد که بعضی از مقادیر زیر صفر، نمایش داده شوند.

Softmax: معمولا برای توابع خروجی چند کلاسه استفاده می‌شود.

Sigmoid: معمولا برای توابع خروجی دو کلاسه استفاده می‌شود.

با توجه به این توضیحات، لایه دوم باید از softmax استفاده کند و لایه اول باید یکی از relu وleaky relu را استفاده کند.

Relu:

Test Accuracy: 0.980400025844574

Test Loss: 0.06649816036224365

Validation Accuracy: 0.9825999736785889

Validation Loss: 0.0633833035826683

Train Accuracy: 0.9987000226974487

Train Loss: 0.008613154292106628

Leaky relu:

Test Accuracy: 0.9746999740600586

Test Loss: 0.08215195685625076

Validation Accuracy: 0.977400004863739

Validation Loss: 0.07940288633108139

Train Accuracy: 0.9940000176429749

Train Loss: 0.021806929260492325

در اینجا متوجه می‌شویم که معرفی دیتاهای منفی، به درد دیتاست mnist نمی‌خورد. علت این‌ امر این است که ذات mnist با این اتفاق در تداخل است.

# تاثیر dropout

یک تکنیک تنظیم‌گری است که به طور معمول در شبکه‌های عصبی استفاده می‌شود. این تکنیک با حذف تصادفی درصد معینی از نورون‌ها در طول آموزش، به جلوگیری از overfitting کمک می‌کند.

انتخاب تصادفی: در طول آموزش، برای هر نمونه آموزشی، زیرمجموعه‌ای تصادفی از نورون‌ها در هر لایه (به جز لایه‌های ورودی و خروجی) با احتمال از پیش تعریف‌شده (مثلاً 20%، 50%) حذف می‌شوند.

غیرفعال‌سازی نورون‌ها: نورون‌های انتخاب‌شده برای آن تکرار آموزشی خاص غیرفعال می‌شوند. اتصالات ورودی و خروجی آن‌ها نیز در طول پاس‌های مستقیم و معکوس فرآیند آموزش نادیده گرفته می‌شوند.

یادگیری با شبکه کاهش‌یافت: نورون‌های باقی‌مانده از ساختار شبکه اصلاح‌شده یاد می‌گیرند. از آنجایی که نورون‌های مختلف در هر تکرار حذف می‌شوند، شبکه به طور مؤثر انواع مختلفی از خود را در طول آموزش یاد می‌گیرد.

model.add(tf.keras.layers.Dense(1024, activation='relu', input\_shape=(28 \* 28,)))

model.add(tf.keras.layers.Dropout(0.1))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

بدون dropout:

Test Accuracy: 0.980400025844574

Test Loss: 0.06649816036224365

Validation Accuracy: 0.9825999736785889

Validation Loss: 0.0633833035826683

Train Accuracy: 0.9987000226974487

Train Loss: 0.008613154292106628

۱۰ درصد:

Test Accuracy: 0.9815999865531921

Test Loss: 0.061822447925806046

Validation Accuracy: 0.9810000061988831

Validation Loss: 0.06562712043523788

Train Accuracy: 0.9980999827384949

Train Loss: 0.010288889519870281

۲۰ درصد:

Test Accuracy: 0.9807000160217285

Test Loss: 0.06399271637201309

Validation Accuracy: 0.9818000197410583

Validation Loss: 0.06364921480417252

Train Accuracy: 0.9977399706840515

Train Loss: 0.011165808886289597

۳۰ درصد:

Test Accuracy: 0.9800000190734863

Test Loss: 0.06396471709012985

Validation Accuracy: 0.982200026512146

Validation Loss: 0.05994435399770737

Train Accuracy: 0.9976599812507629

Train Loss: 0.012121105566620827

۴۰ درصد:

Test Accuracy: 0.9814000129699707

Test Loss: 0.059860944747924805

Validation Accuracy: 0.9814000129699707

Validation Loss: 0.0618421696126461

Train Accuracy: 0.9967799782752991

Train Loss: 0.015245678834617138

مشاهده می‌شود که بهترین dropout ، برابر با ۱۰ درصد است.

# تاثیر batch normalization

این روش به مشکلی به نام "جابجایی همپایه داخلی" (internal covariate shift) رسیدگی می کند و مزایای زیادی را به همراه دارد:

سرعت بخشیدن به آموزش: با استانداردسازی فعال سازی ها، BatchNorm فرآیند آموزش را تثبیت می کند و به شبکه اجازه می‌دهد تا با سرعت بیشتری و با دامنه وسیع‌تری از نرخ یادگیری آموزش ببیند.

کاهش جابجایی همپایه داخلی: استانداردسازی اثر جابجایی همپایه داخلی را کاهش می دهد و یادگیری ویژگی های معنادار را برای لایه های بعدی آسان تر می کند.

بهبود تنظیم (Regularization): BatchNorm می تواند به عنوان یک تنظیم کننده عمل کند، وابستگی شبکه را به وزن ها و فعال سازی های خاص کاهش دهد و به طور بالقوه به جلوگیری از بیش برازش (overfitting) کمک کند.

بدون batch normalization

Test Accuracy: 0.9817000031471252

Test Loss: 0.05697740614414215

Validation Accuracy: 0.9817000031471252

Validation Loss: 0.05857020244002342

Train Accuracy: 0.9965599775314331

Train Loss: 0.01473208237439394

پس از batch normalization

Test Accuracy: 0.9814000129699707

Test Loss: 0.059853728860616684

Validation Accuracy: 0.9825999736785889

Validation Loss: 0.06220643222332001

Train Accuracy: 0.997219979763031

Train Loss: 0.01094264816492796

تفاوت بسیار زیادی بین این دو وجود ندارد ولی با این حال بنظر می‌رسد که بدون batch normalization نیز این مدل به خوبی کار کند. علت این امر می‌تواند این باشد که تا الان پردازش زیادی روی این مدل شده است و با توجه به این‌که دیتاست mnist نسبتا ساده است، ممکن است با اضافه کردن batch normalization به overfitting برسیم.