شبکه های عصبی عمیق معمولا دارای چندین میلیون پارامتر قابل یادگیری هستند. تعداد زیاد پارامتر ها می تواند به دو صورت مشکل ساز شود:

\* با افزایش تعداد پارامتر ها زمان آموزش به تبع افزایش پیدا میکند.

\* در صورت موجود نبودن داده کافی برای آموزش ، یا به دقت خوبی دست پیدا نمیکنید یا مدل شما بیش برارزش خواهد شد.

پست امروز به این امر اختصاص دارد \*\*کاهش زمان آموزش \*\* / \*\* افزایش دقت شبکه\*\* . بدین منظور دو مقاله که اخیرا منتشر شده اند را مورد بحث و بررسی قرار خواهیم داد.

# مقاله اول

مقاله اول [Snapshot Ensembles: Train 1, get M for free]( <https://arxiv.org/abs/1704.00109)است،اما> چرا این مقاله؟ اکثر شرکت کننده گان در رقابت های مختلف Kaggle مخصوصا رقابت هایی که مربوط به داده عکس میشود،برای بالا بردن دقت خروجی از تکنیکی به نام **Model ensembling استفاده میکنند . این مدل معمولا باعث بالابردن 3 الی 7 درصدی دقت خروجی میشود که بین این شرکت کنندگان از محبوبیت بالایی برخوردار است.**

این تکنیک در واقع به مجموعه ای از مدل ها اشاره دارد که به صورت گروهی کار میکنند . ایده به این صورت است که تعدادی مدل با ابر پارامتر های متفاوت را بر روی دیتای مورد نظر آموزش داده میشود و از میانگین این مدل ها برای پیش بینی استفاده میشود. همانطور که در عکس زیر نمایش داده شده است این روشها میتوانند میانگین گیری و یا استفاده از رای اکثریت باشند. تنوع مدل ها باعث میشود که نوعی از واریانس در عملکرد کلی مدل نهایی ما به وجود آید که توزیع خطا حول هدف متمرکز شده و با افزایش نمونه گیری از این توزیع به نتایج بهتری میرسیم. جزییات بیشتر در این لینک (<http://www.scholarpedia.org/article/Ensemble_learning>) وجود است. در صورتی که با روش ترکیب دسته بندی کننده ها در kaggle آشنایی ندارید این لینک (https://mlwave.com/kaggle-ensembling-guide/) مطالب مفیدی را در اختیار شما خواهد گذاشت.

![\_config.yml]({{ site.baseurl }}/images/8/1.gif)

رد پای این تکنیک را میتوانید در مقاله های زیادی ببینید به طور مثال این مقاله که Sate of the art رقابت تشخیص حالت چهره موجود است . از سه مدل مختلف برای کلاسه بندی داده های استفاده کرد . نتایج هر یک از مدل ها در جدول زیر آورده شده است:

![\_config.yml]({{ site.baseurl }}/images/8/2.png)

دقت هر یک از مدل ها بین 71 تا 72 است . نویسندگان مقاله برای افزایش دقت خروجی از ترکیب سه مدل استفاده کردند که دقت نهایی به 75.2 افزایش پیدا کرد!

اما مشکل این تکنیک چیست؟

ترکیب مدل های مختلف هر چند باعث افزایش دقت نهایی میشود،اما ما نمیتوانیم بر روی یک مدل برای پیش بینی خروجی تکیه کنیم، از طرفی آموزش چندین مدل مختلف طاقت فرسا است مخصوصا اگر مدل شما عمیق، حجم داده ها زیاد و توان پردازشی سیستم شما پایین باشد!

Stochastic Gradient Descent

بگذارید ابتدا کمی در مورد طبیعت این الگوریتم صحبت کنیم. یکی از مهمترین پارامتر های این الگوریتم نرخ یادگیری است.اگر نرخ یادگیری بسیار زیاد باشد SGD نقاط minima را نادیده میگیرد.از طرفی اگر نرخ یادگیری کم باشد این الگوریتم در یک بهینه محلی گیر می افتد(هر چند با افزایش نرخ یادگیری این امکان وجود دارد که الگوریتم از این نقاط خارج شود.)

نویسندگان این مقاله از قابل کنترل این الگوریتم،یعنی گیر افتادن در بهینه محلی و خارج شدن از آن به شکلی متفاوت استفاده میکنند. هر کدام از این نقاط دارای نرخ خطای مشابهی هستند اما خطاهایی که توسط آنها تولید میشود دارای میژگی های متفاوتی میباشند.

به شکلی که در مقاله آورده شده است دقت کنید:

![\_config.yml]({{ site.baseurl }}/images/8/3.png)

در شکل سمت چپ الگوریتم SGD  استاندارد نمایش داده شده است که سعی دارد بهترین نقطه کمینه را پیدا کند . شکل سمت راست الگوریتم SGD  نمایش داده شده است که در یک نقطه محلی گیر می کند و مجددا از آن خارج میشود فرآیندی که به صورت تکراری رخ میدهد. با این تکنیک اکنون ما سه نقطه کمینه داریم (که در شکل با سه برچسب 1و2و3 نمایش داده شده اند)که هر گدام از آنها دارای نرخ خطای مشابه،اما با ویژگی های متفاوتی هستند.

در این مقاله از ویژگی این نقاط برای ایجاد چندین مدل برای پیش بینی نهایی استفاده شده است.به عبارتی در هر زمان که الگوریتم SGD به یک نقطه محلی میرسد یک کپی از مدل ذخیره میشود، که در انتها بخشی از مدل ترکیبی ما خواهد بود.

در این مقاله به جای اینکه به صورت تلاش کنیک که چه هنگام به نقطه مینیمم رسیده ایم و چه هنگام از آن خارج شویم، آنها از تابعی استفاده میکنند که این فرآیند را به صورت خود کار انجام میدهد روشی که Cyclic Cosine Annealing نام دارد.

![\_config.yml]({{ site.baseurl }}/images/8/4.png)

در این فرمول α نرخ یادگیری جدید و α0 نرخ یادگیری قبلی است.T تعداد کل تکرار آموزش و M تعداد Snapshot هایی است که ما میخواییم ذخیره کینم .قسمت بالای کسر همانند یک تابع یکنواخت کاهشی عمل میکند.

در ریاضیات یکنواختی (یک نوایی) یک تابع به ما میگوید که تابع افزایش یا کاهش پیدا میکند. به عنوان مثال تابع زیر را در نظر بگیرید:

g(x)=x^3

اگر گراف ایت باع را رسم کنیم g در تمامی نقاط افزایش پیدا میکند در نتیجه این تابع یک تابع یکنواخت (monotonic ) است. (ییشتر بخوانید (<https://en.wikipedia.org/wiki/Monotonic_function)>)

![\_config.yml]({{ site.baseurl }}/images/8/5.jpg)

![\_config.yml]({{ site.baseurl }}/images/8/6.png)

در شکل بالا هر خط نقطه چین عمودی به یک Snapshot از مدل اشاره دارد. بعد 300 گام آموزش تعداد 6 مدل ذخیره میشود. در شکل بالا توجه کنید که خطا چگونه به تدریجی قبل از ذخیره هر Snapshot ذخیره میگردد.این به این علت است که نرخ یادگیری به صورت پیوسته کاهش پیدا میکند. بعد از هر Snapshot نرخ یادگیری به حالت اولیه باز گردانده میشود این امر باعث میشود که مسیر گرادیان از نقطه کمینه بازگردانده شود و به تدریج به یک نقطه محلی دیگر سوق داده شود.(برای جزییات بیشتر بخش 4.2(section 4.2) الگوریتم SGD در شبکه های عصبی(http://leon.bottou.org/publications/pdf/nimes-1991.pdf) را مطالعه کنید)

با این استراتژی بدون افزایش هزینه آموزش، دقت را افزایش میدهیم.در شکل زیر نمونه ای از نتایج نمایش داده شده است.

![\_config.yml]({{ site.baseurl }}/images/8/7.png)

کد این مقاله در این لینک(**https://github.com/titu1994/Snapshot-Ensembles**

) موجود است و روش استفاده از آن به آسانی در شبه کد ذیر توضح داده شده است:

from snapshot import SnapshotCallbackBuilder

M = 5 # number of snapshots

nb\_epoch = T = 200 # تعداد گام آموزش

alpha\_zero = 0.1 # در این مقاله نرخ یادگیری اولیه 0.1 است

model\_prefix = 'Model\_'

snapshot = SnapshotCallbackBuilder(T, M, alpha\_zero)

...

model = Sequential() OR model = Model(ip, output) # Some model that has been compiled

model.fit(trainX, trainY, callbacks=snapshot.get\_callbacks(model\_prefix=model\_prefix))

#مقاله دوم: افزایش سرعت آموزش با Freeze کردن به تدریج لایه ها (<https://arxiv.org/pdf/1706.04983.pdf>)

نویسندگان این مقاله متدی را پیشنهاد میدهند که سرعت یادگیری را Freeze کردن لایه ها افزایش میدهند. راههای متفاوتی بدین منظور در این مقاله شرح داده شده است که باعث افزایش سرعت یادگیری و افزایش جزیی دقت میشود.

#اما Freeze کردن لایه ها به چه معنی است؟

Freeze کردن یک لایه به این معنی است که مقادیر وزنها در آن لایه به صورت ثابت باقی میمانند . این تکنیک به Transfer leaning معروف است.قبل از پرداختن به مقاله شاید صحبت کردن در این زمینه خالی از لطف نباشد!

در عمل تعداد کمی از محققین داده یک شبکه کانولوشن(با مقدار دهی تصادفی اولیه) را از اول آموزش میدهند زیرا جمع آوری داده های زیاد در اندازه مناسب وقت گیر است ، در عوض رایج این است که از مدل های از قبل آموزش داده شده استفاده کرد بصورتی که از لایه های ابتدایی این مدل ها برای استخراج ویژگی ها استفاده میشود به این عملیات Transfer Learning گفته میشود. سه سناریو عمده انتقال آموزش وجود دارد:

1. ConvNet as fixed feature extractor . در این حالت ما یک مدل که قبلا بر روی دیتابیسی مثل Imagenet آموزش داده شده است را انتخاب میکنیم و آخرین لایه تماما متصل آن را حذف میکنیم (Imagenet دارای 1000 کلاس است و لایه خروجی حاوی امتیازات کلاس ها است ) و بقیه لایه ها را به عنوان لایه های استخراج ویژگی به صورت ثابت باقی میمانند.به عنوان مثال در Alexnet آخرین لایه قبل از فعال ساز Sofmax یک بردار 4096 بعدی است این لایه حاوی ویزگی هایی است که CNN codes نامیده میشوند. عکس زیر از لکچر 7، کورس یادگیری عمیق بهار 2017 استندفورد استخراج شده است( بیشتر بخوانیدhttp://cs231n.stanford.edu/slides/2017/cs231n\_2017\_lecture7.pdf)

![\_config.yml]({{ site.baseurl }}/images/8/8.png)

1. Fine-tuning the ConvNet . دومین استراتژی آموزش مجدد تمامی یا بخشی از مدل توسط دیتا بیس جدید است . به این صورت که ابتدا مقادیر شبکه بارگذاری میشوند و مجددا توسط الگوریتم پس انتشار آموزش داده میشود . در عمل فقط لایه های بالای شبکه مجددا آموزش داده میشوند به دلیل اینکه لایه های ابتدایی در واقع ویژگی های عکس مثل لبه ها و ... را شناسایی میکنند که تقریبا برای هر داده تصویر میتواند مفید باشند اما لایه های موجود در سطح بالاتر ویژگی های پیچیده تری را آموزش میبینند که بیشتر مختص دیتابیس اصلی است. به عنوان مثال در دیتابیس Imagenet حاوی تعداد زیادی نژاد سگ است در نتیجه بخش مهمی از شبکه به ویژگی های نژاد سگ ها و تفاوت آنها اختصاص داده شده است . به عنوان مثال در صورتی که از یک شبکه از قبل آموزش داده شده بر روی Imagenet برای شرکت در رقابت dogs vs cats (https://www.kaggle.com/c/dogs-vs-cats) استفاده شود به نتایج بسیار خوبی دست پیدا میکنیم (دقت بالای 98 درصد) .در صورتی که تعداد نمونه ها در دیتا بیس جدید زیاد باشد معمولا میتوان چندین لایه انتهایی را آموزش داد اما در صورت وجود داده کم فقط لایه آخر (کلاسه بند خطی) را آموزش میدهند .
2. Pretrained models . از آنجایی که آموزش شبکه های عصبی کانولوشن مدرن بین دو تا سه هفته با استفاده از چندین کارت گرافیک انجام میشوند بیشتر محققن علوم داده مدل آموزش داده شده خود را در اختیار سایر کاربران قرار میدهند تا دیگران از آنها سود ببرند و برای کلاسه بندی های متفاوت از آنها استفاده کنند.به عنوان مثال کتابخانه Caffe محلی به نام Model Zoo ( <https://github.com/BVLC/caffe/wiki/Model-Zoo>) دارد جایی که محققیق مدل های آموزش داده شده را به اشتراک میگذارند.

هنگام که یک لایه Freeze میشود عملیات اصلاح وزن ها دیگر انجام نمیشود و این امر باعث افزایش چشمگیر سرعت آموزش میشود، به عبارتی در صورتی که نیمی از شبکه Freeze شود سرعت آموزش نسب به آموزش یک مدل کامل به نصف کاهش پیدا میکند از سوی دیگر، شما نیاز دارید که مدل را آموزش دهید ، اگر شما بخشی از لایه ها را خیلی زود freeze کنید، مدل نتایج ها را به صورت ناصحیح پیش بینی خواهد کرد.

در این مقاله نویسندگان یک راه برای freeze کردن لایه ها پیشنهاد میدهند که باعث کاهش فرآیند آموزش میشود. در ابتدا تمام شبکه مانند یک شبکه معمولی قابل آموزش است، بعد از چند گام اولین لایه Freeze میشود و مابقی شبکه به آموزش ادامه میدهند، بعد از چند گام لایه دوم Freeze میشود و این عملیات به همین صورت ادامه پیدا میکند. به جای ثابت نگه داشتن نرخ یادگیری در تمامی مدل، نرخ یادگیری به صورت لایه به لایه با توجه به معادله زیر تغییر میکند.

![\_config.yml]({{ site.baseurl }}/images/8/9.png)

ئر این معادله **α نرخ یادگیری، t تعداد** iteration و iبه لایه iام و **αi(0) مقدار اولیه نرخ یادگیری است. نویسندگام مقاله نرخ اولیه در هر لایه را افزایش میدهند در نتیجه هر لایه به یک اندازه آموزش داده میشوند.**

**به دلیل اینکه آموزش لایه اول زودتر از بقیه لایه ها متوقف میشود این لایه نسبت به دیگر لایه ها برای زمان کمتری آموزش داده میشود ، برای جبران نرخ یادگیری برای هر لایه افزایش داده میشود.لایه هایی که در انتها قرار گرفته اند به دلیل اینکه مدت زمان بیشتری آموزش داده میشوند نرخ یادگیری کمتری دارند.**

![\_config.yml]({{ site.baseurl }}/images/8/10.png)

**متد ارایه شده در این مقاله باعث افزایش 20 درصدی سرعت آموزش و افزایش 3 درصدی دقت خروجی برای** DenseNets(<https://arxiv.org/abs/1608.06993>)، افزایش 20 درصدی سرعت آموزش برای ResNet(<https://arxiv.org/abs/1512.03385)بدون> تاثیر در دقت شد اما بر روی مدل VGG (<https://arxiv.org/abs/1409.1556)هیچ> تاثیری نداشت.