دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

درس مبانی هوش محاسباتی

پروژه اول: پیاده سازی شبکه‌های عصبی

جواب بخش سوالات تشریحی

نگارش:

مهدی رحمانی

۹۷۳۱۷۰۱

استاد درس:

دکتر عبادزاده

بهار ۱۴۰۱

بخش اول

سوال دوم)

با نرمال سازی دسته‌ای(Batch Normalization) به جای اینکه فقط یک بار قبل از اعمال شبکه عصبی نرمال سازی کنیم، خروجی هر سطح نرمال سازی میشود و در ورودی سطح بعد از آن استفاده میشود که باعث افزایش سرعت همگرایی و درنتیجه آموزش میشود.

یکی از رایج ترین مشکلات Overfitting است. زمانی که مدل در داده های آموزشی بسیار خوب عمل میکند اما قادر به پیش بینی دقیق داده های تست نیست. دلیلش Overfittingاست. یک راه چنین مشکلی Regularization است. تکنیک‌های Regularization به بهبود مدل کمک می‌کند و به آن امکان می‌دهد سریع‌تر همگرا شود. چندین ابزار منظم سازی در انتهای خود داریم، برخی از آنها dropout ، early stopping ، weight initialization techniques و batch normalization هستند.

نرمال سازی دسته ای، فرآیندی برای سریعتر و پایدارتر کردن شبکه های عصبی از طریق افزودن لایه های اضافی در یک شبکه عصبی عمیق میباشد. لایه جدید عملیات استانداردسازی و نرمال سازی را روی ورودی لایه‌ای که از لایه قبلی می آید انجام می دهد. درواقع نرمال‌سازی دسته‌ای روشی است که activationها را روی mini-batch با سایز مشخص نرمالسازی می‌کند. برای هر ویژگی، نرمال سازی دسته‌ای، میانگین و واریانس آن ویژگی را در mini-batch محاسبه می کند. سپس میانگین را کم می کند و ویژگی را بر انحراف استاندارد mini-batch تقسیم می کند.

اما دلیل دسته‌ای بودن در نرمال سازی دسته ای این است که یک شبکه عصبی معمولی با استفاده از مجموعه ای از داده های ورودی به نام دسته (Batch) آموزش داده می شود. به طور مشابه، فرآیند نرمال سازی در نرمال سازی دسته ای به صورت دسته ای انجام می شود، نه روی یک ورودی. همچنین یک فرایند دو مرحله‌ای است که اول ورودی نرمال سازی میشود و سپس rescaling و offsetting انجام میشوند.

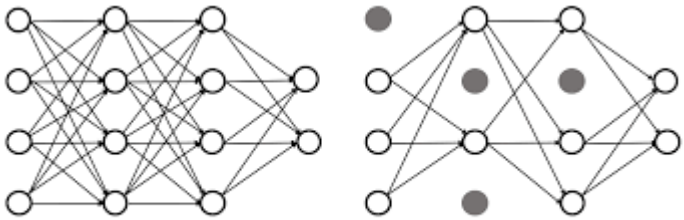
از دلایل اصلی استفاده میتوان به ۱) افزایش سرعت آموزش و ۲) هندل کردن internal covariate shift اشاره کرد.

در کدنویسی عملی، Batch Normalization را بعد از تابع فعالیت یا قبل از تابع فعالیت در یک لایه اضافه می کنیم. اغلب محققان نتایج خوبی را در پیاده سازی Batch Normalization پس از تابع فعالیت یافته اند. باتوجه به نمونه کدهای سایت‌های ۱و ۲ میتوان گفت که میتوان آن‌ها را بین هر دولایه قرار داد. البته اینکه چه تعداد و کجا قرار دهیم به برنامه نویس ربط دارد. از لینک‌های ۳ و ۴ نیز کمک گرفتم.

سوال سوم)

طبق این روش تعدادی از نورون‌های شبکه عصبی به صورت رندوم در حین آموزش به صورت موقت حذف یا به عبارتی خاموش میشوند و تاثیری در مدل نمیگذارند. به صورت دقیق تر در این روش به ازای همه نورون‌ها به جز نورون‌های لایه آخر، عددی تصادفی بین ۰ و ۱ تولید میشود و سپس اگر این عدد تصادفی متناظر با هر نورون از یک حد آستانه مثلا ۰٫۵ کمتر بود، آن نورون و connectionها و سیناپس‌هایش را از شبکه حذف میکنیم و در فرآیند آموزش شرکت نمیکنند. همچنین دلیل اینکه برای لایه آخر این کار را نمیکنیم این است که نمیخواهیم یک وقت یکی از نورون‌های این لایه حذف شود و درواقع یکی از خروجی‌هامون مثلا همیشه ۰ باشد.

به این ترتیب شبکه ما ساده‌تر و کوچک‌تر میشود و بنابراین مدلی هم که یادمیگرد خیلی مثل قبل پیچیده نخواهد بود و احتمال overfit شدن کاهش میابد. درواقع به این ترتیب نورون‌های به یک نورون ورودی خاص که لایه قبل آن هاست، وابسته نمیشوند چراکه ممکن است به صورت رندوم حذف شود و این باعث کاهش بایاس شدن به ورودی ها میشود. همچنین جزئیات ورودی‌ها یادگرفته نمیشوند و فقط اطلاعات مهم ذخیره میشوند و باعث میشود شبکه از اطلاعات مفید برای پیش بینی استفاده کند.



(a) Standard Neural Network. (b) Network applying dropout

حال داخل کد میتوان drop out را به صورت layers.Dropout(p) اضافه کرد که p احتمال است و مقدار بین ۰ و ۱ دارد و با اون احتمال تعدادی نورون را خاموش میکند.

طبق توضیحات این لینک، Dropout را می توان بعد از لایه های کانولوشن (مانند Conv2D) و بعد از لایه های pooling(مانند MaxPooling2D) استفاده کرد. اغلب، dropout فقط بعد از لایه‌های pooling استفاده می‌شود، اما به صورت کلی در دست طراح و برنامه نویس است و به خلاقیت او بستگی دارد. ما هم طبق توضیحات و مثل فیلم آموزشی بعد از لایه‌ها pooling و همچنین بعداز دوتا از لایه‌های fully connected به جز لایه اخر میگذاریم. همچنین طبق توضیحات این سایت dropout را بعد از batchNorm گذاشتم.

سوال چهارم)

طبق سایت keras ، Optimizerهای زیر در دسترس هستند:

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

حال به توضیح مختصر هریک در ادامه میپردازیم.

گرادیان کاهشی تصادفی (SGD)

گرادیان کاهشی تصادفی یکی از گونه‌های گرادیان کاهشی است. این گرادیان، پارامترهای مدل را به طور پیوسته به‌روزرسانی می‌کند. در این روش، پارامترهای مدل پس از محاسبه زیان در هر نمونه آموزش تغییر می‌یابند. بنابراین، اگر مجموعه‌داده ۱۰۰۰ ردیف داشته باشد، گرادیان کاهشی تصادفی پارامترهای مدل را ۱۰۰۰ بار در یک چرخه از مجموعه‌داده به‌روزرسانی می‌کند؛ برخلاف گرادیان کاهشی که عمل به‌روزرسانی فقط یک بار در آن انجام می‌شد.

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right)$$

بر این اساس، $\{x(i), y(i)\}$نمونه‌های آموزشی هستند.

از آنجایی که پارامترهای مدل به طور مرتب به‌روزرسانی می‌شوند، پارامترها واریانس و نوسان بالایی در توابع زیان دارد؛ شدت آن فرق می‌کند.

مزایا:

۱. به‌روزرسانی پیوسته‌ی پارامترهای مدل؛ نتیجه: همگرایی در زمان کمتر
۲. نیاز به حافظه کمتر زیرا ذخیره‌سازیِ مقادیر توابع زیان الزامی نیست.
۳. احتمال بدست آوردن کمینه‌ی جدید

معایب:

۱. واریانس بالا در پارمترهای مدل
۲. احتمال ادامه فعالیت حتی پس از بدست آوردن کمینه کلی.
۳. برای اینکه همگراییِ شبیه به گرادیان کاهشی به دست آید، مقدار نرخ یادگیری به آرامی کاهش می‌یابد.

RMSprop

RMSprop یک تکنیک بهینه سازی مبتنی بر گرادیان است که در آموزش شبکه های عصبی استفاده می شود. این پیشنهاد توسط پدر back-propagation، جفری هینتون ارائه شد. گرادیان توابع بسیار پیچیده مانند شبکه‌های عصبی تمایل به ناپدید شدن یا انفجار دارند، زیرا داده‌ها از طریق تابع منتشر می‌شوند (به مسئله vanishing gradient اشاره دارد). Rmsprop به عنوان یک تکنیک تصادفی برای یادگیری mini-batch توسعه داده شد.

RMSprop با استفاده از میانگین متحرک مجذور گرادیان ها برای نرمال کردن گرادیان، به موضوع فوق می پردازد. این نرمال‌سازی اندازه گام (تکانه) را متعادل می‌کند، گام را برای شیب‌های بزرگ برای جلوگیری از انفجار کاهش می‌دهد و برای شیب‌های کوچک برای جلوگیری از ناپدید شدن، پله را افزایش می‌دهد. به زبان ساده، RMSprop از نرخ یادگیری تطبیقی به جای گرفتن نرخ یادگیری به عنوان یک هایپرپارامتر استفاده می کند. این بدان معناست که نرخ یادگیری در طول زمان تغییر می کند.

نحوه آپدیت شدن در RMSprop به صورت زیر است:

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

Adam

روش تخمین تکانه تطبیق‌پذیر (Adam) با تکانه‌های مرتبه اول و دوم کار می‌کند. Adamیک الگوریتم مبتنی بر گرادیان مرتبه اول از توابع هدف تصادفی است که بر اساس برآوردهای تطبیقی گشتاورهای مرتبه پایین‌تر است. Adam یکی از جدیدترین الگوریتم‌های بهینه‌سازی پیشرفته است که توسط بسیاری از متخصصان یادگیری ماشین استفاده می‌شود. تکانه اول که توسط تکانه دوم به روز میشود، جهت به روز رسانی را مشخص میکند.

قانون به روز رسانی آدام به صورت زیر است:

$$\theta_{n+1} = \theta_n - \frac{\alpha}{\sqrt{\hat{v}_n} + \epsilon}\hat{m}_n$$

الگوریتم آن نیز در عکس زیر آمده:

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

مزایا:

۱. این روش بسیار سریع است و به سرعت همگرایی پیدا می‌کند.
۲. نرخ یادگیری و واریانس بالا را اصلاح می‌کند.

معایب:

۱. هزینه‌های محاسباتی بالا

AdaDelta

روش AdaDelta بسطی از روش AdaGrad است که هدف آن کمتر کردن کاهش فزاینده و یکنواخت کردن نرخ یادگیری در این روش است. به جای تجمع مربعات همه‌ی گرادیان‌های قبلی، روش AdaDelta تعداد مربعات جمع شده را به تعداد محدودی مثل w محدود می‌کند. به جای ذخیره‌ی غیربهینه‌ی wتا از مربعات گرادیان‌ها، جمع گرادیان به صورت بازگشتی، میانگین میراشونده‌ی همه‌ی مربعات گرادیان‌های قبلی تعریف می‌شود. میانگین در لحظه‌ی t یعنی E[g2]t فقط (با یک کسر γ که شبیه به ثابت شتاب است) به میانگین قبلی و گرادیان فعلی بستگی دارد:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

مزایا:

حال، نرخ یادگیری کاهش نمی‌یابد و آموزش متوقف نمی‌شود.

معایب:

نیازمند محاسبات زیاد

روش گرادیان انطباقی (AdaGrad)

یکی از معایب برخی بهینه‌سازها ، این است که نرخ یادگیری در همه پارامترها و چرخه‌ها ثابت است. روش فعلی، نرخ یادگیری η را برای هر پارامتر و هر بازه زمانی تغییر می‌دهد و یک الگوریتم بهینه‌سازی مرتبه دوم به شمار می‌آید که با مشتق تابع خطا کار می‌کند.

مشتق تابع خطا پارامترهای مشخص در زمان مشخص t

$$g_{t.i} = \nabla_{\theta_t} J(\theta_{t.i})$$

به‌روزرسانی پارامترها با ورودی مشخص i و زمان t

$$\theta_{t+1.i} = \theta_{t.i} - \frac{\eta}{\sqrt{G_{t.ii} + \epsilon}} \cdot g_{t.i}$$

در اینجا، η نرخ یادگیری است که بر اساس گرادیان‌های پیشین، در پارامتر $\theta(i)$ و زمان مشخص تغییر می‌یابد. هر درایه‌ی قطری i,i جمع مربعات گرادیان‌ها نسبت به θi در گام زمانی t است و ϵ یک ضریب هموارسازی است که از ایجاد صفر در مخرج جلوگیری می‌کند (معمولا در مقیاس 10^{-8} است). جالب است که بدون جذر گرفتن، الگوریتم عملکرد بسیار ضعیف‌تری دارد.

مزایا:

۱. تغییر نرخ یادگیری در هر پارامتر آموزش
۲. عدم نیاز به تنظیم دستیِ نرخ یادگیری
۳. امکان آموزش با داده‌های پراکنده

معایب:

۱. نیاز به محاسبات سنگین در هنگام محاسبه مشتق مرتبه دوم
۲. کاهش نرخ یادگیری در پی آموزش کُند

Adamax

الگوریتم AdaMax توسعه ای برای الگوریتم بهینه سازی تخمین حرکت تطبیقی (Adam) است. به طور گسترده تر، توسعه ای برای الگوریتم Gradient Descent Optimization است.

به طور کلی، AdaMax به طور خودکار اندازه گام جداگانه (نرخ یادگیری) را برای هر پارامتر در مسئله بهینه سازی تطبیق می دهد.

Nadam

Nesterov-accelerated Adaptive Moment Estimation یا همان Nadam بسطی است برای الگوریتم بهینه‌سازی Adam برای افزودن گرادیان شتاب‌دار نستروف (NAG) یا تکانه نستروف، که نوع بهبود یافته‌ای از تکانه است. این مومنتوم یک ترم نمایی از میانگین گرادیان (مومنتوم اول) را به االگوریتم گرادیان کاهشی اضافه میکند. این باعث هموارتر شدن تابع noisy میباشد و باعث میشود که همگرایی سریعتر رخ دهد.

Ftrl

Follow The Regularized Leader یا (FTRL) یک الگوریتم بهینه سازی است که در گوگل برای پیش بینی نرخ کلیک در اوایل دهه ۲۰۱۰ توسعه یافته است. برای مدل‌های کم عمق با فضای حالت sparse و بزرگ مناسب است

از لینک های ۱ و ۲ و ۳ و ۴ کمک گرفته شد.

در اینجا ما از SGD و RMSprop و Adagrad کمک گرفتیم و آموزش دادیم.

سوال پنجم)

معیار precision :

حداکثر مقدار این معیار یک ویا ۱۰۰ درصد و حداقل مقدار آن صفر است و هرچه مواردی که برنامه به **غلط پیش بینی کرده است** که به آن False Positive می‌گوییم نسبت به پیش بینی‌های درست یا True Positive بیشتر باشد مقدار Precisionکمتر خواهد شد. درواقع میگوید، وقتی که مدل نتیجه را مثبت (positive) پیش‌بینی می‌کند، این نتیجه تا چه اندازه درست است؟

در فرمول زیر TP مخفف True Positive و FP مخفف False Positive است.

$$Prcision = \frac{TP}{TP + FP}$$

زمانی که ارزش false positives بالا باشد، معیار صحت، معیار مناسبی خواهد بود. فرض کنید، مدلی برای تشخیص سرطان داشته باشیم و این مدل Precision پایینی داشته باشد. نتیجه این امر این است که این مدل، بیماری بسیاری از افراد را به اشتباه سرطانی تشخیص می‌دهد. نتیجه این امر استرس زیاد، آزمایش‌های فراوان و هزینه‌های گزافی را برای بیمار به دنبال خواهد داشت.

معیار Recall :

حداکثر مقدار این معیار یک ویا ۱۰۰ درصد و حداقل مقدار آن صفر است و هرچه مواردی که ما انتظار داشتیم پیش بینی شوند ولی برنامه پیش بینی نکرده‌است که به آن False Negative می‌گوییم نسبت به پیش بینی‌های درست یا True Positive بیشتر باشد مقدار Recall کمتر خواهد شد. در فرمول زیر TP مخفف True Positiveو FN مخفف False Negative است.

$$Recall = \frac{TP}{TP + FN}$$

زمانی که ارزش false negatives بالا باشد، معیارRecall ، معیار مناسبی خواهد بود. فرض کنیم مدلی برای تشخیص بیماری کشنده ابولا داشته باشیم. اگر این مدل Recall پایینی داشته باشد چه اتفاقی خواهد افتاد؟ این مدل افراد زیادی که آلوده به این بیماری کشنده هستند را سالم در نظر می‌گیرد و این فاجعه است.

معیار F1-score :

زمانی که می‌خواهید معیار ارزیابی شما میانگینی از دو مورد قبلی باشد یعنی همان Recall یا Precision می‌توانید از میانگین هارمونیک این دو معیار استفاده کنید که به آن **معیار f1-score** می‌گویند. فرمول آن به صورت زیر است:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

معیارF1 ، یک معیار مناسب برای ارزیابی دقت یک آزمایش است. این معیار Precision و Recall را با هم در نظر می‌گیرد. معیار F1 در بهترین حالت، یک و در بدترین حالت صفر است.

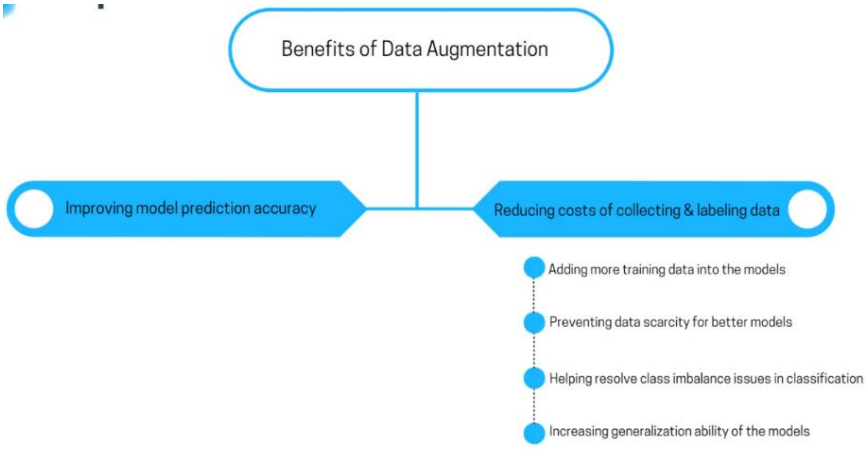
بهترین مدل به دست آمده با Optimizer به نام Adam میباشد. برای آن خروجی‌ها نشان داده شده اند.

بخش دوم

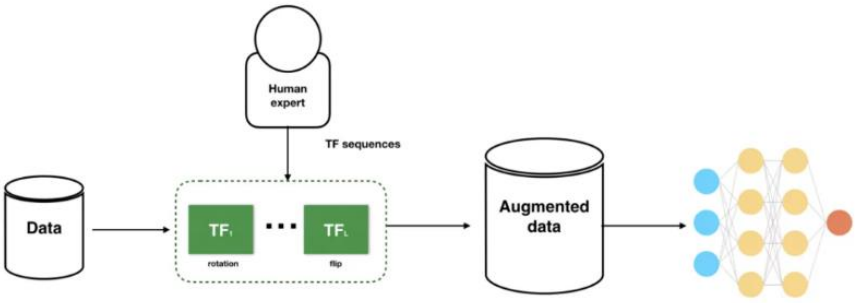
سوال اول)

Data Augmentation در تجزیه و تحلیل داده ها تکنیک هایی هستند که برای افزایش حجم داده ها از روش‌هایی مثل افزودن کپی هایی که در اثر کمی تغییر در داده های موجود، یا داده های مصنوعی جدیدی که از داده های موجود ایجاد شدند، استفاده میکند. پس این داده های جدید ممکن است با تغییر اندک در داده های قبلی یا با استفاده از الگوریتم های deep تر تولید شوند. این به عنوان یک منظم کننده عمل می کند و به کاهش Overfitting در هنگام آموزش یک مدل یادگیری ماشینی کمک می کند. در تجزیه و تحلیل داده ها ارتباط نزدیکی با Oversampling دارد.

از فواید آن میتوان به موارد زیر اشاره کرد:



نحوه کارکرد آن به صورت شماتیک در تصویر زیر آمده است:

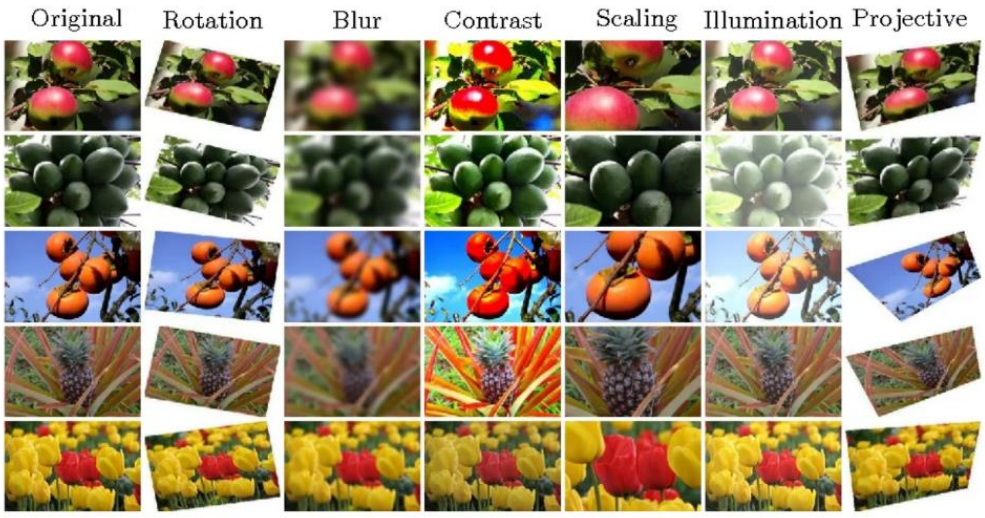


در image classification و segmentation داریم:

برای افزایش داده ها، ایجاد تغییرات ساده در داده های بصری رایج است. علاوه بر این، شبکه‌های متخاصم مولد (GAN) برای ایجاد داده‌های مصنوعی جدید استفاده می‌شوند. فعالیت های کلاسیک پردازش تصویر برای Data Augmentation عبارتند از:

- padding
- random rotating
- re-scaling,
- vertical and horizontal flipping
- translation (image is moved along X, Y direction)
- cropping
- zooming
- darkening & brightening/color modification
- grayscaling
- changing contrast
- adding noise
- random erasing

برای مثال عکس‌های زیر با این روش‌ها تولید شدند:



مدل‌های پیشرفته تر برای Data Augmentation عبارتست از:

- Adversarial training/Adversarial machine learning
- Generative adversarial networks (GANs)
- Neural style transfer
- Reinforcement learning

در حوزه NLP داریم:

افزایش داده در حوزه NLP به اندازه حوزه بینایی کامپیوتر محبوب نیست. افزایش داده های متنی به دلیل پیچیدگی یک زبان دشوار است. روش های رایج برای افزایش داده در NLP عبارتند از:

- عملیات افزایش آسان داده ها (EDA): جایگزینی مترادف، درج کلمه، تعویض کلمه و حذف کلمه
- ترجمه برگشتی: ترجمه مجدد متن از زبان مقصد به زبان اصلی آن
- جاسازی کلمات متنی

مدل‌های تشخیص تصویر و NLP معمولاً از روش‌های تقویت داده‌ها استفاده می‌کنند. همچنین، حوزه تصویربرداری پزشکی از تقویت داده ها برای اعمال تبدیل بر روی تصاویر و ایجاد تنوع در مجموعه داده ها استفاده می کند. دلایل علاقه به افزایش داده ها در حوزه‌های درمانی و پزشکی :

- مجموعه داده کوچک برای تصاویر پزشکی
- به دلیل قوانین حفظ حریم خصوصی داده های بیمار، به اشتراک گذاری داده ها آسان نیست
- تنها تعداد کمی از بیماران وجود دارند که از داده های آنها می توان به عنوان داده های آموزشی در تشخیص بیماری های نادر استفاده کرد

باتوجه به توضیحات داده شده، برای زمانی که داده برای آموزش مدل کم است یا مشکل Overfitting داریم مثلا میتوان از Data augmentation کمک گرفت. بنابراین در این موارد روی دیتاهای train اعمال میشود و مرسوم نیست که روی دیتای تست اعمال کنیم اگرچه اگر این تکنیک ها را روی دیتاهای تست اعمال کنیم میتوانیم دیتاهای بیشتری برای تست کردن مدلمان داشته باشیم. در این لینک هم به این موضوع اشاره شده است. همچنین در این لینک نیز توضیحات بیشتر آورده شده است.