بسم الله الرحمن الرحیم

دانشکده ریاضی و علوم کامپیوتر

پایان‌نامه برای دریافت درجه کارشناسی ارشد در رشته علوم کامپیوتر

گرایش علوم تصمیم و دانش

شبکه مولد رقابتی وَسِراِستین

استادان راهنما

دکتر مهدی زعفرانیه و دکتر محمود امین‌طوسی

استاد مشاور

دکتر امین رفیعی

پژوهشگر:

سیده افسانه صالحی ساداتی

شهریور ۱۳۹۹

سوگند نامه دانش آموختگان دانشگاه حکیم سبزواری

به نام خداوند جان و خرد کزین برتر اندیشه بر نگذرد

اینک که به خواست آفریدگار پاک، کوشش خویش و بهره گیری از دانش استادان و سرمایه‌های مادی و معنوی این مرز و بوم، توشه‌ای از دانش و خرد گردآورده‌ام، در پیشگاه خداوند بزرگ سوگند یاد می‌کنم که در به کارگیری دانش خویش، همواره بر راه راست ودرست گام بردارم. خداوند بزرگ، شما شاهدان، دانشجویان و دیگر حاضران را به عنوان داورانی امین گواه می گیرم که ازهمه دانش و توان خود برای گسترش مرزهای دانش بهره گیرم و از هیچ کوششی برای تبدیل جهان به جایی بهتر برای زیستن، دریغ نورزم. پیمان می‌بندم که همواره کرامت انسانی را در نظر داشته باشم و همنوعان خود را در هر زمان و مکان تا سر حد امکان یاری دهم. سوگند می خورم که در به کارگیری دانش خویش به کاری که با راه و رسم انسانی، آیین پرهیزگاری، شرافت و اصول اخلاقی برخاسته از ادیان بزرگ الهی، به ویژه دین مبین اسلام، مباینت دارد دست نیازم. همچنین در سایه اصول جهان شمول انسانی و اسلامی، پیمان می‌بندم از هیچ کوششی برای آبادانی و سرافرازی میهن و هم میهنانم فروگذاری نکنم و خداوند بزرگ را به یاری طلبم تا همواره در پیشگاه او و در برابر وجدان بیدار خویش و ملت سرافراز، بر این پیمان تا ابد استوار بمانم.

نام و نام خانوادگی: سیده افسانه صالحی ساداتی

تاریخ و امضا:

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب سیده افسانه صالحی ساداتی به شماره دانشجویی ۹۷۱۳۱۸۵۰۱۵ دانشجوی رشته علوم کامپیوتر مقطع تحصیلی کارشناسی ارشد تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه برداری شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی صلاح (اعم از اداری و قضایی) به عهده ی اینجانب خواهد بود و دانشگاه هیچ گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: سیده افسانه صالحی ساداتی

تاریخ و امضا:

مجوز بهره برداری از پایان‌نامه

بهره برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

☐ بهره برداری از این پایان‌نامه برای همگان بلامانع است.

☐ بهره برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.

☐ بهره برداری از این پایان‌نامه تا تاریخ ..... ممنوع است.

استادان راهنما: دکتر مهدی زعفرانیه

دکتر محمود امین‌طوسی

تاریخ و امضا:

تقدیم به:

پدر و مادرم

سپاس خداوندگار حکیم را که با لطف بی کران خود، آدمی را زیور عقل آراست.

در آغاز وظیفه خود می دانم از زحمات بی دریغ استادان راهنمای خود، جناب آقای دکتر مهدی زعفرانیه و جناب آقای دکتر محمود امین طوسی، صمیمانه تشکر و قدردانی کنم که قطعاً بدون راهنمایی های ارزنده ایشان، این مجموعه به انجام نمی رسید.

از جناب آقای دکتر امین رفیعی که زحمت مطالعه و مشاوره این رساله را تقبل فرمودند و در آماده سازی این رساله، به نحو احسن اینجانب را مورد راهنمایی قرار دادند، کمال امتنان را دارم.

در پایان، بوسه می زنم بر دستان خداوندگاران مهر و مهربانی، پدر و مادر عزیزم و بعد از خدا، ستایش می‌کنم وجود مقدس شان را و تشکر می‌کنم از خانواده عزیزم به پاس عاطفه سرشار و گرمای امیدبخش وجودشان، که بهترین پشتیبان من بودند.

سیده افسانه صالحی ساداتی

شهریور ۱۳۹۹

فهرست مطالب

فهرست جداول

فهرست تصاویر

فهرست الگوریتم‌ها

فرم چکیده ی پایان‌نامه ی دوره ی تحصیلات تکمیلی

مدیریت تحصیلات تکمیلی

دانشگاه حکیم سبزواری

نام خانوادگی دانشجو:صالحی ساداتی	نام:سیده افسانه	ش. دانشجویی:۹۷۱۳۱۸۵۰۱۵
استادان راهنما: دکتر مهدی زعفرانیه و دکتر محمود امین‌طوسی		
استاد مشاور: دکتر امین رفیعی		
دانشکده ریاضی و علوم کامپیوتر	رشته: علوم کامپیوتر	گرایش: علوم تصمیم و دانش
مقطع: کارشناسی ارشد	تاریخ دفاع:شهریور ۱۳۹۹	تعداد صفحات: ۹۰
عنوان پایان‌نامه: شبکه مولد رقابتی وَسِراستین		
کلید واژه‌ها:یادگیری عمیق، شبکه عصبی، شبکه مولد رقابتی، نظریه بازی		

چکیده: شبکه‌های مولد رقابتی نوعی سیستم یادگیر برای تولید نمونه‌های جدید از داده‌های آموزشی است. این روش از رقابت دو شبکه عصبی در یک نظریه بازی به تولید نمونه‌های جدید می‌پردازد. هدف این شبکه ها تولید نمونه های جدید با توزیع های پیچیده همانند تصاویر، است. برای ساخت این نمونه های جدید از یک توزیع ساده نویز تصادفی شروع کرده و با استفاده از شبکه عصبی، آن را به توزیع پیچیده مورد نظر می رساند. در این پایان نامه با ساختار شبکه عصبی و شبکه های مولد رقابتی آشنا می شویم و نمونه‌های تولید شده از این شبکه را مورد بررسی قرار می دهیم.

پیش‌گفتار

شبکه های عصبی مصنوعی نوعی از سیستم های دینامیکی هستند که با پردازش داده های تجربی، دانش و قانون داده ها را به ساختار شبکه منتقل می کنند و توانایی یادگیری قوانین کلی بر اساس محاسبات داده های عددی و مثال ها را دارند، از این رو به سیستم های هوشمند معروف هستند. شبکه‌های مولد رقابتی (شبکه مولد متخاصم) [۱] نوعی سیستم یادگیر برای تولید نمونه‌های جدید از داده‌های آموزشی است. این روش ترکیبی از شبکه‌های عصبی و استفاده از نظریه بازی [۲] به عنوان یک روش شناسایی است و به تولید نمونه های جدید می‌پردازد. دو شبکه عصبی به کار برده شده در این شبکه ها، مولد و متمایز کننده نام دارند. با آموزش این شبکه‌ها توسط الگوریتم پس‌انتشار خطا [۳] به تولید نمونه های جدید از توزیع های پیچیده با ابعاد بالا دست می‌یابند. شبکه مولد رقابتی برای تولید تصاویر، ویدیو، متن، موسیقی و اثر انگشت جدید مورد استفاده قرار می‌گیرند. همچنین برای افزایش وضوح تصویر، بازسازی تصاویر قدیمی و اثر انگشت‌های موجود نیز کاربرد دارند. شبکه GAN برای ساخت این نمونه های جدید از یک توزیع ساده نویز تصادفی شروع کرده و با تبدیلات و توابع مختلف، آن را به توزیع پیچیده مورد نظر می رساند. راه مستقیمی برای یادگیری تبدیلات یا توابع وجود ندارد و روش معمول استفاده از شبکه عصبی است.

این پایان نامه شامل ۳ فصل است:

در فصل ۱ مفاهیم اولیه در مورد شبکه عصبی، نحوه‌ی عملکرد آن، توابع مورد نیاز و شبکه عصبی پیچشی مطرح خواهد شد.

در فصل ۲ به شبکه‌های مولد رقابتی، نحوه‌ی آموزش و انواع آن، نظریه بازی و ارتباط آن با شبکه مولد رقابتی پرداخته می‌شود.

در فصل ۳ شبکه مولد رقابتی وَسِراِستین [۴] مطرح شده و روش آموزش آن به همراه الگوریتم متناظر مورد بررسی قرار می‌گیرد.

---

[۱] Generative Adversaial Network(GAN) [۲] Game Theory [۳] Backpropagation Algorithm

[۴] Wasserstein Generative Adversial Network

مطالب این پایان‌نامه برگرفته از کتاب‌ها و مقالات زیر می‌باشد:

1. Chollet, Francois. Deep Learning with Python. Manning,November 2017.

2. Vasilev, I., Slater, D., Spacagna, G., Roelants, P., and Zocca, V. Python Deep Learning: Explorirg deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow, 2nd Edition. Packt Publishing, 2019.

3. Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein generative adversarial networks. in Precup, Doina and Teh, Yee Whye, eds. , Proceedings of the 34th International Conference on Machine Learning, vol. 70 of Proceedings of Machine Learning Research, pp. 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

نمادگذاری ریاضی

در این پایان‌نامه برای تعریف متغیرها و پارامترها از یک نمادگذاری واحد استفاده می‌شود. به این منظور شیوه مراجع [۱، ۲] ملاک عمل قرار گرفته است.

شبکه عصبی . با حروف بزرگ و برجسته [۱] به صورت $\mathbf{X}$ نشان داده می‌شود.

ماتریس . با حروف بزرگ همانند زیر نشان داده می‌شود.

$$X = \begin{bmatrix} \mathrm{x}_1^T \\ \mathrm{x}_2^T \\ \vdots \\ \mathrm{x}_n^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix} \in \mathbb{R}^{n \times m}$$

بردار . با حروف کوچک و ایستاده [۲] همانند زیر نشان داده می‌شود.

$$\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

نمادهای زیر در این پایان نامه استفاده می‌شوند.

$\mathbf{J}$. تابع هزینه شبکه عصبی

$\mathbf{V}$. تابع هزینه شبکه مولد رقابتی

---

[۱] bold [۲] Roman

$\mathbf{b}$. بردار بایاس [1]

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

θ . پارامتر (وزن) شبکه

θ_g . وزن‌های شبکه مولد

$\mathbf{t}$. خروجی واقعی نمونه‌ها

$\mathbf{y}$. خروجی پیش‌بینی شده نمونه‌ها

$\mathbf{N}$. تعداد نمونه‌های آموزشی

$\mathbf{n}$. تعداد نورون‌های لایه اول

$\mathbf{m}$. تعداد نورون‌های لایه پنهان

$\mathbf{k}$. تعداد نورون‌های لایه خروجی

$\mathbf{G}$. شبکه عصبی مولد

$\mathbf{D}$. شبکه عصبی متمایزکننده

P . توزیع احتمال نمونه‌ها

P_{data} . توزیع احتمال نمونه‌های آموزشی

P_g . توزیع احتمال نمونه‌های تولید شده توسط مولد

$\mathbf{z}$. بردار اولیه تصادفی برای ورودی شبکه مولد

ρ . فاصله‌ی میان دو توزیع احتمالی

$\mathbf{W}$. فاصله‌ی وَسِراِستین

---

[1] Bias

فصل ۱

شبکه عصبی و یادگیری عمیق

مغز انسان به عنوان سیستم پردازش اطلاعات با ساختار موازی، دارای ویژگی‌های شگفت‌انگیز است و پیاده‌سازی این ویژگی‌ها در یک سیستم مصنوعی از دیرباز تاکنون از هدف‌های والای بشر بوده است. شبکه‌های عصبی [۲] تا حدودی به این هدف رسیده‌اند. آن‌ها برای مسائلی که راه‌حلی ندارند و یا به راحتی قابل حل نیستند، مورد استفاده قرار می‌گیرد. این شبکه‌ها با پردازش روی داده‌های موجود، به اطلاعات و قانون نهفته در داده‌ها پی می‌برند [۳].

یادگیری عمیق [۳] یک ساختار شبکه عصبی با تعداد لایه های زیاد و یکی از رو به رشدترین زیرحوزه های یادگیری ماشین [۴] است. مدل های یادگیری ماشین می توانند فضای نهفته آماری تصاویر، موسیقی و ویدیو را بیاموزند و سپس می توانند از این فضا نمونه بگیرند و آثار هنری جدیدی با خصوصیات مشابه مدل هایی که در داده های آموزشی خود دیده اند، ایجاد کنند [۴].

در برنامه نویسی های کلاسیک داده ها و قوانین به عنوان ورودی به شبکه داده می‌شود و طبق قوانین موجود، داده‌ها تغییر کرده و خروجی تولید می‌شود، اما در یادگیری عمیق داده ها و جواب مسئله به شبکه داده می‌شود تا قوانین و ویژگی‌های موجود در داده‌ها را به دست آورد. به این روند استخراج ویژگی [۵] گویند که یکی از مهمترین مباحث در یادگیری ماشین است. یادگیری عمیق برخلاف مدل های ساده شبکه عصبی، به صورت خودکار ویژگی ها را استخراج می کند [۴].

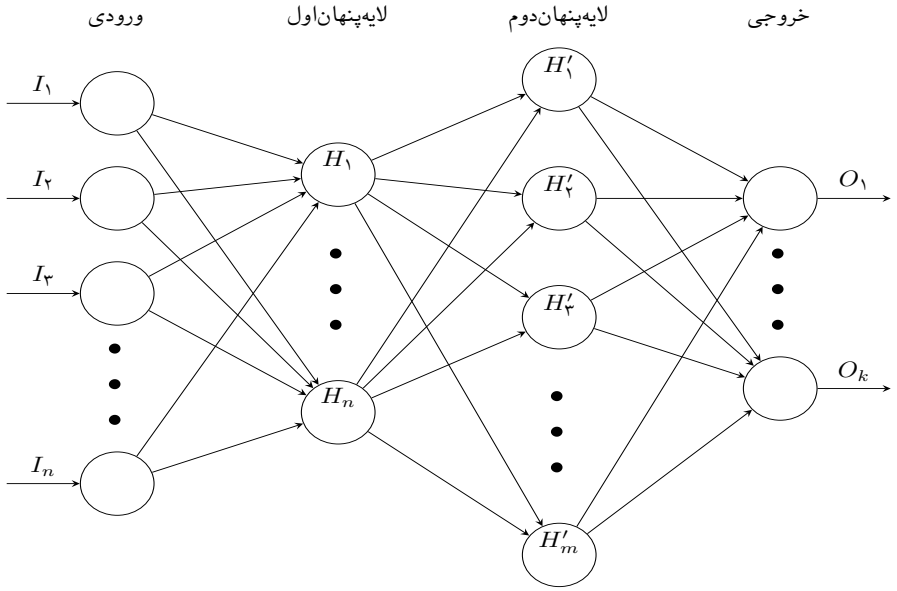
این فصل دربرگیرنده اهمیت شبکه عصبی و توضیح ساختار شبکه عصبی و توابع مورد استفاده در آن است. همچنین نحوه‌ی عملکرد قاعده پس‌انتشار خطا در شبکه عصبی و الگوریتم این قاعده ذکر شده است، و مفهوم تنسورها، شبکه عصبی پیچشی و برخی از لایه‌های آن بیان شده است.

---

[۲] Neural Network [۳] Deep Learning [۴] Machine Learning [۵] Feature Engineering

۱-۱ ساختار شبکه عصبی

شبکه عصبی همانند مغز انسان متشکل از نورون‌ها و اتصال بین آن‌هاست. نورون کوچکترین واحد پردازش اطلاعات است [۳]. همچون شکل ۱-۱ شبکه عصبی از یک لایه ورودی و یک لایه خروجی و چندین لایه میانی تشکیل شده است که در هر لایه تعداد مشخصی نورون وجود دارد. همه‌ی نورون‌های یک لایه به تمام نورون‌های لایه بعد خود متصل است. جریان اطلاعات نورون‌ها یک‌طرفه است، از سمت ورودی شروع، به لایه‌های پنهان و در نهایت به لایه خروجی می‌رسد [۵] . به این نوع شبکه عصبی ایستا یا پیشخور [۱] می‌گویند. در مقابل اگر مسیر اطلاعات علاوه بر مسیر اصلی، از سمت لایه خروجی یا لایه پنهان به خود لایه یا لایه ورودی باشد، شبکه عصبی بازگشتی یا پسخور [۲] گویند. این نوع مدل را مدل شبکه پویا گویند [۳]. نورون‌های ورودی شبکه دارای وزن‌های اولیه θ هستند. مقادیر وزن‌ها در تمام لایه‌ها به همراه مقدار بایاس b، پارامترهای شبکه را تشکیل می‌دهند، شبکه عصبی در پی یافتن بهترین وزن است که در بخش ۱-۱-۳ به بررسی آن خواهیم پرداخت. برای جلوگیری از صفر شدن وزن‌های جدید و انعطاف‌پذیر شدن تابع فعالیت، مقدار ثابت بدون وزن بایاس در هر لایه به شبکه اضافه می‌شود. نورون‌های ورودی شبکه با داشتن وزن‌های اولیه θ و بایاس b وارد لایه اول شده و در تابع فعالیت آن لایه قرار گرفته می‌شود و خروجی این لایه، ورودی لایه بعدی خواهد بود. جریان اطلاعات از سمت ورودی به خروجی است [۱]. به همین ترتیب هر لایه تابع فعالیت مخصوص خود را دارد که در بخش ۱-۱-۱ به انواع تابع فعالیت و ویژگی‌های آن می‌پردازیم.

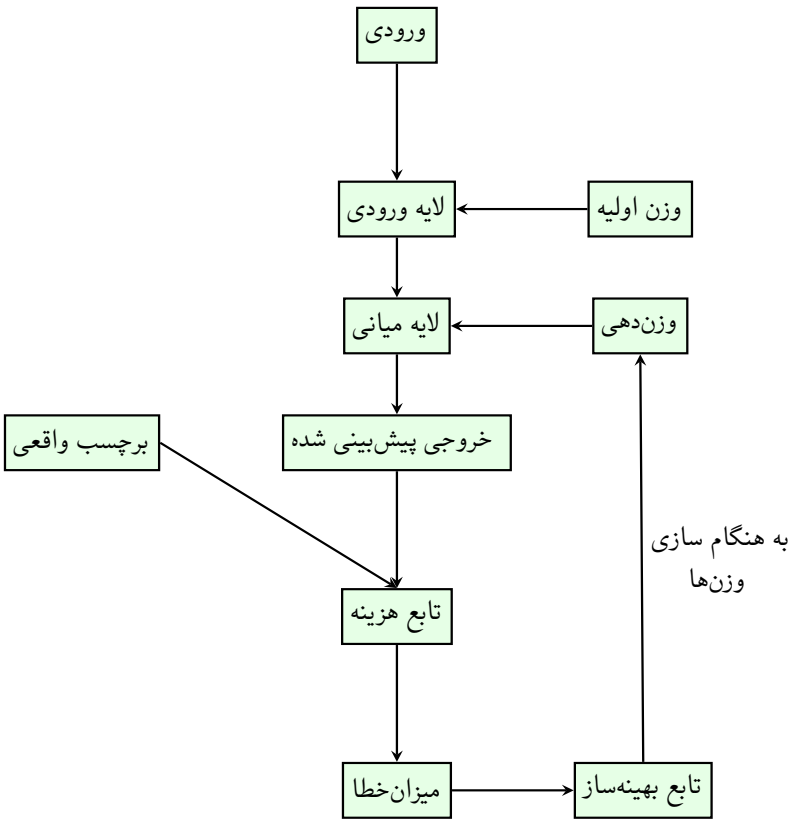


شکل ۱-۱: شمای کلی یک شبکه عصبی با دو لایه پنهان

در شبکه‌های عصبی نیاز به معیاری برای تشخیص میزان خطای شبکه وجود دارد. این معیار تحت عنوان تابع

[۱] feed-forward network [۲] feed-back network

هزینه (بخش ۱-۱-۲)، با مقایسه تفاوت میان مقدار واقعی و خروجی شبکه، میزان خطای شبکه عصبی را بیان می‌کند [۴]. هرچه اختلاف میان مقدار واقعی و خروجی شبکه، کم باشد، شبکه کارایی بهتری داشته است. میزان خطای به دست آمده توسط تابع هزینه، به تابع بهینه ساز داده می‌شود. این تابع طبق توضیحات بخش ۱-۱-۳، وزن‌های نورون‌ها را به گونه‌ای تغییر می‌دهد که در مراحل بعدی اجرا، میزان خطا کاهش پیدا کند. شکل ۱-۲ نمایانگر روند کار شبکه عصبی است. این روند ادامه پیدا می‌کند تا وزن بهینه شبکه به دست آید.



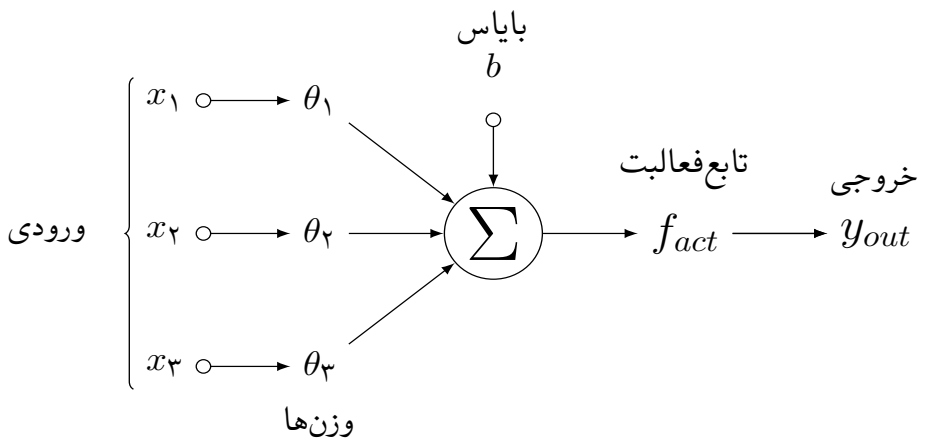
شکل ۱-۲: روند اجرای شبکه عصبی

۱-۱-۱ تابع فعالیت

شبکه عصبی توانایی تمرکز روی ویژگی‌های یک ورودی دلخواه و تولید خروجی مناسب از آن ورودی را دارد. تابع فعالیت(تابع محرک) [۱] برای فعال سازی نورون‌ها و تبدیل آن‌ها به یک سیگنال خروجی مناسب استفاده می شود [۵]. تابع فعالیت یک عملیات غیرخطی است که از خطی بودن ترکیب نورون‌ها و تبدیل شبکه عصبی به یک مدل خطی ساده، جلوگیری می‌کند [۱]. در هر لایه‌ی شبکه عصبی به صورت مجزا، یک تابع فعالیت موجود می‌باشد

[۱] Activation function

که ورودی آن، نورون x و وزن‌های θ است و خروجی این تابع، به عنوان ورودی لایه بعد خواهد بود. تابع فعالیت انواع مختلفی دارد که برخی در جدول ۱-۱ موجود می‌باشد [۳، ۵، ۱].



شکل ۱-۳: تابع فعالیت در یک لایه

همانند شکل ۱-۳ در هر لایه، حاصل ضرب نورون‌های $x_۱, x_۲, \ldots, x_n$ آن لایه با وزن‌های $\theta_۱, \theta_۲, \ldots, \theta_n$ به همراه یک بایاس b به صورت زیر به تابع فعالیت داده می‌شود [۶، ۱].

$$\sum_{i=۱}^{n} x_i\theta_i + b = x_۱\theta_۱ + x_۲\theta_۲ + \cdots + x_n\theta_n + b \qquad (۱-۱)$$

با در نظر گرفتن یک ورودی به عنوان ضریبی برای بایاس که برابر با ۱ باشد، و تغییر نماد بایاس به $\theta_۰$ رابطه بالا به صورت زیر به تابع فعالیت داده می‌شود:

$$\begin{aligned}\sum_{i=۰}^{n} x_i\theta_i &= \theta_۰ + x_۱\theta_۱ + x_۲\theta_۲ + \cdots + x_n\theta_n \\ &= \begin{bmatrix} ۱ & x_۱ & x_۲ & \ldots & x_n \end{bmatrix} \begin{bmatrix} \theta_۰ \\ \theta_۱ \\ \vdots \\ \theta_n \end{bmatrix}\end{aligned} \qquad (۲-۱)$$

جواب تابع فعالیت غیرخطی f، برداری مانند $\mathbf{y}$ است که به صورت زیر به دست می‌آید [۱]:

$$\mathbf{y} = f\left(\sum_{i=۱}^{n} x_i\theta_i + b\right) = f\left(\theta^T X + \mathrm{b}\right) \qquad (۳-۱)$$

یا به عبارتی

$$\mathbf{y} = f\left(\sum_{i=۰}^{n} x_i\theta_i\right) = f\left(\theta^T X + \theta_۰\right) \qquad (۴-۱)$$

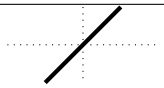
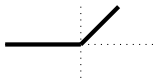
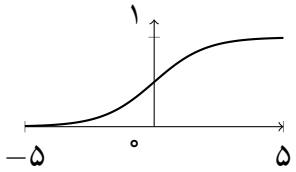
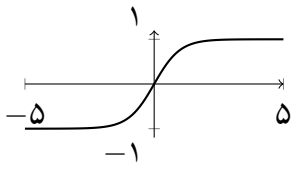
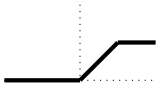
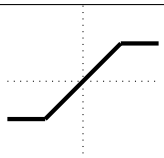
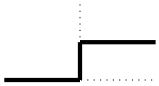
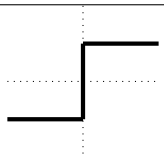
در حالت کلی برای محاسبه تمام خروجی‌های یک لایه با در نظر گرفتن اندیس i برای تعداد ورودی و اندیس j برای تعداد خروجی، رابطه زیر مطرح می‌شود.

$$\mathbf{y}_j = f\left(\sum_{i=۱}^{n} x_i\theta_{i,j} + b\right) \qquad (۵-۱)$$

در این رابطه تعداد بایاس و تعداد ستون‌های ماتریس وزن با تعداد نورون‌های خروجی هر لایه، برابر و تعداد سطرهای ماتریس وزن با تعداد نورون‌های ورودی لایه، برابر است. با در نظر گرفتن مقدار $\mathbf{y}$ برابر صفر، رابطه ابرصفحه جداکننده‌ی نمونه‌های ورودی به دست می‌آید. این رابطه را مرز تصمیم‌گیری گویند که نمونه‌های ورودی را طبقه‌بندی می‌کند. مرز تصمیم‌گیری در فضای یک بعدی داده‌ها، یک نقطه است، در فضای دوبعدی داده‌ها، خط راست است، در فضای سه بعدی، یک صفحه و در ابعاد بالاتر ابرصفحه خواهد بود و در حالت کلی به صورت زیر بیان می‌شود [۳، ۱].

$$\sum_{i=۱}^{n} \theta_i x_i + b = ۰ \qquad (۶-۱)$$

جدول ۱-۱: انواع توابع فعالیت

ردیف	نام	تعریف تابع	نمودار
۱	هویت (identity)	$f(x) = x$	
۲	یک‌سو ساز (Relu)	$f(x) = \begin{cases} ۰ & \text{if } x < ۰ \\ x & \text{if } x \geq ۰ \end{cases}$	
۳	سیگموید (sigmoid)	$f(x) = \frac{۱}{۱ + e^{-x}}$	
۴	تانژانت هیپربولیک	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	
۵	آستانه‌ای خطی	$f(x) = \begin{cases} ۰ & \text{if } x < ۰ \\ x & \text{if } ۰ \leq x \leq ۱ \\ ۱ & \text{if } x > ۱ \end{cases}$	
۶	آستانه‌ای خطی متقارن	$f(x) = \begin{cases} -۱ & \text{if } x < -۱ \\ x & \text{if } -۱ \leq x \leq ۱ \\ ۱ & \text{if } x > ۱ \end{cases}$	
۷	آستانه‌ای دو مقداره	$f(x) = \begin{cases} ۰ & \text{if } x < ۰ \\ ۱ & \text{if } x \geq ۰ \end{cases}$	
۸	آستانه‌ای دو مقداره متقارن	$f(x) = \begin{cases} -۱ & \text{if } x < ۰ \\ ۱ & \text{if } x \geq ۰ \end{cases}$	

۱-۱-۲ تابع هزینه

در ساخت شبکه عصبی، پارامترهایی همچون تابع هزینه [۱]، تابع بهینه سازی [۲] و تابعی برای سنجش دقت، مورد استفاده قرار می‌گیرد. تابع هزینه روشی برای ارزیابی کارکرد مدل است. این تابع تفاوت بین جواب واقعی که از قبل در پایگاه داده داریم، و جواب پیش بینی شده مدل شبکه عصبی را محاسبه می‌کند. اگر جواب پیش بینی شده از نتایج واقعی منحرف شده باشد، مقدار تابع هزینه زیاد است. به تدریج با کمک الگوریتم پس‌انتشار خطا (بخش ۱-۲) می‌توان این مقدار را کاهش داد. به عبارت دیگر هر چه تفاوت جواب پیش بینی شده و مقدار واقعی کمتر باشد، جواب شبکه به جواب واقعی نزدیک‌تر است و شبکه به درستی پیش بینی انجام می‌دهد. توابع هزینه مختلفی وجود دارد که به برخی از آن‌ها اشاره می‌کنیم. در نظر داشته باشید که $\mathbf{t}$ مقادیر جواب واقعی و $\mathbf{y}$ مقادیر جواب پیش بینی شده است [۴، ۱، ۷]:

- میانگین مربعات خطا (mean squared error)

$$\mathbf{J}(\mathbf{y},\mathbf{t}) = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - t_i)^2 \qquad (۱-۷)$$

- میانگین قدرمطلق خطا (mean absolute error)

$$\mathbf{J}(\mathbf{y},\mathbf{t}) = MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - t_i \mid \qquad (۱-۸)$$

- میانگین خطای ساده (mean bias error)

$$\mathbf{J}(\mathbf{y},\mathbf{t}) = MBE = \frac{1}{n}\sum_{i=1}^{n}(y_i - t_i) \qquad (۱-۹)$$

- اختلاف باینری آنتروپی (binary cross entropy)

$$\mathbf{J}(\mathbf{y},\mathbf{t}) = -\left[t_i \log(y_i) + (1 - t_i)\log(1 - y_i)\right] \qquad (۱-۱۰)$$

---

[۱] Loss Function [۲] Optimizer Function

- اختلاف دسته‌ای آنتروپی (categorical cross entropy)

$$\mathbf{J}(\mathbf{y}, \mathbf{t}) = -\sum_{i=۱}^{n} t_i \log(y_i) \qquad (۱۱-۱)$$

۱-۱-۳ تابع بهینه ساز

در شبکه عصبی، به حداقل رساندن تابع هزینه $\mathbf{J}$ مدنظر است. با پیدا کردن وزن بهینه θ، مقدار تابع $\mathbf{J}$ به حداقل ممکن می‌رسد [۱]. نورونهای شبکه در ابتدا وزن اولیه تصادفی دارند و همانطور که در شکل ۱-۲ نمایان است، در اجراهای بعدی شبکه، این وزنها به وزن‌های جدید و بهتری تغییر داده می‌شوند. تابع بهینه [۱] با مشتق گرفتن از تابع هزینه و به دست آوردن $\nabla \mathbf{J}(\theta)$ ، به جهت مخالف گرادیان تابع ($-\nabla \mathbf{J}(\theta)$) حرکت کرده و وزن جدید را به دست می‌آورد [۴، ۱].

در ابتدا داده‌ها یک وزن اولیه θ دارند و تابع بهینه سازی به ازای هر نمونه، وزن‌ها را به صورت زیر به‌روز می‌کند.

$$\theta_{new} = \theta_{old} + \Delta\theta \qquad (۱۲-۱)$$

$$\Delta\theta = -\eta \nabla \mathbf{J}(\theta) \qquad (۱۳-۱)$$

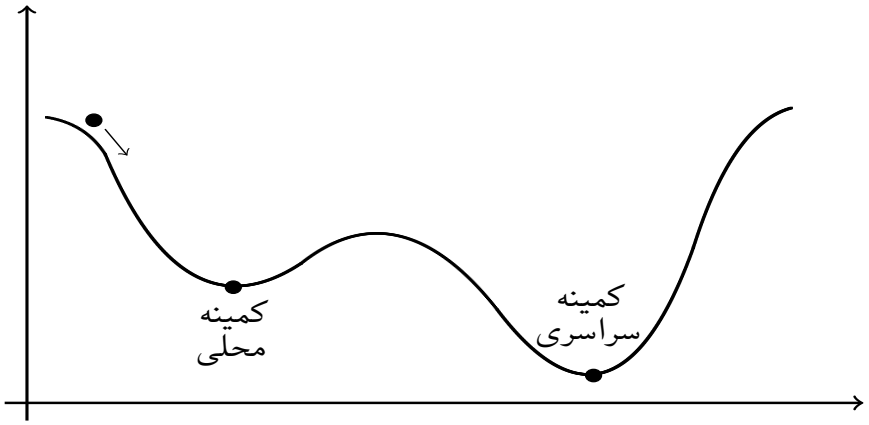
این توابع به دلیل سرعت پایین همانند شکل ۱-۴ در برخی از درّه‌های کمینه محلی متوقف می‌شوند و نمی‌توانند به کمینه سراسری که جواب بهینه است، برسند. برای رفع این مشکل از گشتاور [۲] استفاده می‌شود. گشتاور پارامتر α را به تغییرات وزن اضافه می‌کند. با اضافه شدن پارامتر گشتاور، به روز رسانی وزن‌ها به تغییر وزن قبلی و شیب وزن فعلی وابسته است. در این صورت در قسمت‌های درّه مانند که به کمینه محلی ختم می‌شوند، سرعت افزایش یافته و تابع بهینه از کمینه محلی خارج می‌شود. گشتاور تضمینی برای رسیدن به کمینه سراسری و جواب بهینه است [۴، ۸، ۹]. در این صورت رابطه به صورت زیر تغییر می‌کند :

$$\Delta\theta = -\alpha\Delta\theta - \eta \nabla \mathbf{J}(\theta) \qquad (۱۴-۱)$$

[۱] Optimizer Function [۲] Momentum

و همچنین خواهیم داشت:

$$\theta_{new} = \theta_{old} - \eta \nabla \mathbf{J}(\theta) + \alpha \Delta \theta \qquad (۱۵-۱)$$



شکل ۱-۴: تابع بهینه با استفاده از گشتاور از کمینه محلی با سرعت عبور کرده و به کمینه سراسری می‌رسد.

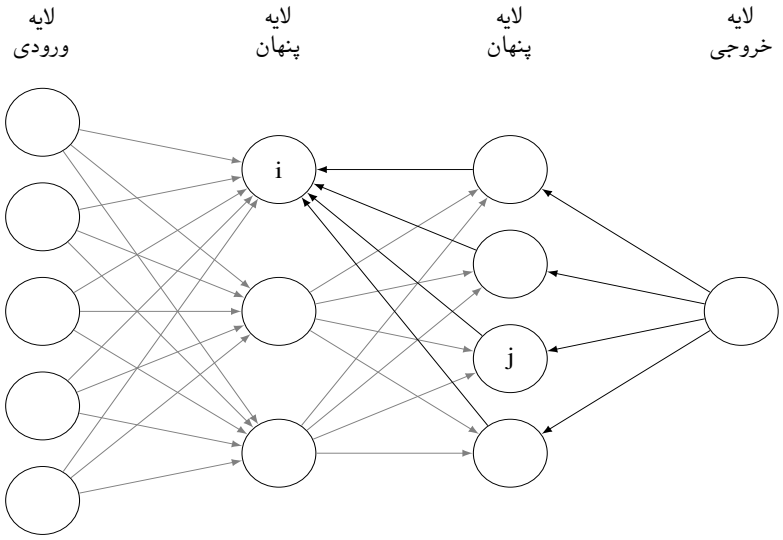
انواع توابع بهینه سازی موجود، با محاسبه بهترین وزن‌ها، مشابه عملیات بیان شده را انجام می‌دهند و متناسب با نوع شبکه عصبی می‌توان از آن‌ها استفاده نمود. از بهینه سازهای معروف می‌توان به روش سریع‌ترین شیب[۱]، سریع‌ترین شیب تصادفی[۲] و همچنین روش‌های توسعه داده شده این روش همانند RMSProp، Adam، Adagrad و Adadelta اشاره کرد [۱۰]. تمام این توابع را میتوان همراه با گشتاور محاسبه کرد [۱۱، ۹]. اکثر این توابع، به خوبی کار می‌کنند و نمی‌توان برتری یکی را نسبت به بقیه بیان کرد، کارایی این توابع در انواع مختلف شبکه عصبی متفاوت است [۱۲].

۱-۲ قاعده پس انتشار خطا

در طول آموزش یک شبکه عصبی میزان خطا با مقایسه مقدار خروجی و مقدار واقعی به دست می‌آید. تغییر وزن‌های شبکه قادر به کاهش میزان خطا در شبکه خواهد بود. در بخش ۱-۱-۳ چگونگی به‌هنگام کردن وزن در شبکه یک لایه، بیان شد. از آن روش در شبکه‌های چند لایه فقط برای به‌هنگام کردن وزن‌های لایه پنهان نهایی استفاده می‌شود؛ زیرا در شبکه‌های چند لایه، مقدار خروجی لایه‌های پنهان در دسترس نیست. برای به‌هنگام کردن وزن‌های لایه‌های پنهان از قاعده پس انتشار خطا [۳] استفاده می‌شود. این قاعده همانند شکل ۱-۵ مراحل شبکه

[۱] Gradient descent (GD) [۲] Stochastic Gradient descent (SGD) [۳] Backpropagation

عصبی را به عقب برمی‌گرداند [۱۳]. قاعده پس انتشار خطا با تخمین میزان خروجی لایه‌های پنهان و محاسبه میزان خطا، وزن‌های لایه‌های پنهان را به گونه‌ای به‌روز می‌کند که خطا به حداقل برسد [۱].



شکل ۱-۵: پس انتشار خطا در شبکه عصبی

۱-۲-۱ تغییر وزن‌های شبکه عصبی با قاعده پس انتشار خطا

قاعده پس انتشار خطا در لایه‌های ورودی، میانی و خروجی مورد استفاده قرار می‌گیرد. دو لایه‌ی L ، $L+1$ را در نظر بگیرید. این دو لایه به صورت متوالی در شبکه عصبی قرار دارند و می‌توانند از جمله لایه‌های ورودی، خروجی و یا پنهان باشند. نورون i در لایه‌ی L و نورون j درلایه‌ی $L+1$ قرار دارد. قاعده پس انتشار خطا برای به‌روز رسانی وزن موجود بین این دو نورون ($\theta_{i,j}$) استفاده می‌شود. در ابتدا فرضیات زیر را در نظر بگیرید [۱، ۱۴]:

$\theta_{i,j}$: وزن موجود بین نورون i از لایه‌ی L و نورون j از لایه‌ی $L+1$

t : مقدار خروجی واقعی

y_i : خروجی لایه‌ی L و همچنین ورودی لایه‌ی$L+1$

y_j :خروجی لایه‌ی $L+1$

y_k :خروجی لایه‌ی $L+2$

در قاعده پس انتشار از تابع هزینه $\mathbf{J}$ برحسب وزن $\theta_{i,j}$ مشتق گرفته می‌شود. طبق مباحث بخش ۱-۱-۲، تابع هزینه تابع صریحی از وزن در لایه‌های پنهان شبکه عصبی نیست. فرض کنید تابع هزینه از طریق مجذور

اختلاف خروجی واقعی و خروجی به دست آمده برای تمام نمونه‌ها به صورت زیر محاسبه شود

$$\mathbf{J} = \frac{1}{n}\sum_{j}(y_j - t_j)^2 \qquad (۱-۱۶)$$

در نتیجه توابع هزینه $\mathbf{J}$ برحسب مقادیر خروجی $\mathbf{y}$ محاسبه می‌شود.

$$y_j = f\left(a_j\right) \qquad (۱-۱۷)$$

مقدار $\mathbf{y}$ برحسب a محاسبه شده و همچنین مقدار a برحسب $\theta_{i,j}$

$$a_j = \sum_{i=1}^{n} y_i \theta_{i,j} \qquad (۱-۱۸)$$

با توجه به روابط بالا برای به دست آوردن مشتق تابع هزینه نسبت به وزن‌ها، می‌بایست از مشتق‌گیری زنجیره‌ای به صورت زیر استفاده نمود.

$$\frac{\partial \mathbf{J}}{\partial \theta_{i,j}} = \frac{\partial \mathbf{J}}{\partial y_j}\,\frac{\partial y_j}{\partial a_j}\,\frac{\partial a_j}{\partial \theta_{i,j}} \qquad (۱-۱۹)$$

طبق رابطه (۱-۱۸) برای به دست آوردن $\frac{\partial a_j}{\partial \theta_{i,j}}$ داریم:

$$\frac{\partial a_j}{\partial \theta_{i,j}} = \frac{\partial \sum_{i=1}^{n} y_i \theta_{i,j}}{\partial \theta_{i,j}} = \sum_{i=1}^{n} \frac{\partial \theta_{i,j}}{\partial \theta_{i,j}} y_i = y_i \qquad (۱-۲۰)$$

مقدار رابطه $\frac{\partial y_j}{\partial a_j}$ با مشتق گرفتن از تابع فعالیت لایه‌ی $L+1$ به دست می‌آید.

$$\frac{\partial y_j}{\partial a_j} = f'\left(a_j\right) \qquad (۱-۲۱)$$

محاسبه مقدار رابطه $\frac{\partial \mathbf{J}}{\partial y_j}$ با مشتق گرفتن از تابع هزینه در لایه‌ی $L+1$ به دست می‌آید($\frac{\partial \mathbf{J}}{\partial y_j} = \mathbf{J}'$). این رابطه برای لایه‌ی خروجی مناسب است؛ زیرا خروجی واقعی و خروجی به دست آمده، مشخص هستند. با محاسبه رابطه‌ی بالا برای لایه‌ی خروجی، به صورت بازگشتی مشتق در لایه‌ی قبلی را می‌توان محاسبه کرد. برای این منظور

از قاعده مشتق‌گیری زنجیره‌ای به صورت زیر استفاده می‌شود.

$$\frac{\partial \mathbf{J}}{\partial y_i} = \sum_j \frac{\partial \mathbf{J}}{\partial y_j} \frac{\partial y_j}{\partial y_i} = \sum_j \frac{\partial \mathbf{J}}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial y_i} \tag{۲۲-۱}$$

در این رابطه مقدار $\frac{\partial y_j}{\partial a_j}$ طبق رابطه (۱-۲۱) محاسبه شده است. با مشتق از رابطه (۱-۱۸) مقدار $\frac{\partial a_j}{\partial y_i} = \theta_{i,j}$ به دست می‌آید. و مقدار $\frac{\partial \mathbf{J}}{\partial y_j}$ با استفاده از لایه‌ی بعدی خود (لایه‌ی $L+2$) به صورت زیر محاسبه می‌شود.

$$\frac{\partial \mathbf{J}}{\partial y_j} = \sum_j \frac{\partial \mathbf{J}}{\partial y_k} \frac{\partial y_k}{\partial y_j} \tag{۲۳-۱}$$

میزان تغییر تابع هزینه نسبت به مقدار تابع فعالیت را به صورت زیر تعریف می‌شود.

$$\delta_j = \frac{\partial \mathbf{J}}{\partial y_j} \frac{\partial y_j}{\partial a_j}$$

این مقدار را به عنوان میزان خطا در نورون y_j در نظر گرفته می‌شود. در این صورت رابطه (۱-۲۲) به صورت زیر بازنویسی می‌شود.

$$\frac{\partial \mathbf{J}}{\partial y_i} = \sum_j \frac{\partial \mathbf{J}}{\partial y_j} \frac{\partial y_j}{\partial y_i} = \sum_j \frac{\partial \mathbf{J}}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial y_i} = \sum_j \delta_j \theta_{i,j} \tag{۲۴-۱}$$

به صورت کلی میزان خطا در نورون y_i در هر لایه‌های پنهان به صورت زیر به دست می‌آید.

$$\delta_i = \left(\sum_j \delta_j \theta_{i,j} \right) \frac{\partial y_i}{\partial a_i} \tag{۲۵-۱}$$

بنابراین طبق رابطه بالا و همچنین رابطه‌ی (۱-۲۰)، می‌توان رابطه (۱-۱۹) را به صورت زیر بازنویسی کرد.

$$\frac{\partial \mathbf{J}}{\partial \theta_{i,j}} = \delta_j \frac{\partial a_j}{\partial \theta_{i,j}} = \delta_j y_i \tag{۲۶-۱}$$

در این صورت به‌هنگام کردن وزن‌ها در لایه‌های میانی شبکه عصبی توسط رابطه زیر خواهد بود.

$$\theta_{i,j} = \theta_{i,j} - \eta \delta_j y_i \tag{۲۷-۱}$$

۱-۲-۲ الگوریتم قاعده پس انتشار خطا

با دانستن فرضیات بیان شده در بخش ۱-۲-۱ الگوریتم قاعده پس انتشار خطا به صورت زیر خواهد بود.

الگوریتم ۱-۱ الگوریتم پس انتشار خطا در یک شبکه عصبی

ورودی: بردار ورودی لایه‌های پنهان $\mathbf{y_i} = \begin{bmatrix} y_{i_0} & y_{i_1} & y_{i_2} & \dots & y_{i_n} \end{bmatrix}^T$،
بردار خروجی تولید شده $\mathbf{y_j} = \begin{bmatrix} y_{j_0} & y_{j_1} & y_{j_2} & \dots & y_{j_k} \end{bmatrix}^T$،
بردار خروجی واقعی $\mathbf{t_j} = \begin{bmatrix} t_{j_0} & t_{j_1} & t_{j_2} & \dots & t_{j_k} \end{bmatrix}^T$
و مقدار η

خروجی: بهترین وزن‌های شبکه عصبی که میزان تابع هزینه را کاهش می‌دهد.

۱: مقدار وزن‌ها و بایاس به صورت تصادفی در نظر گرفته و به شبکه عرضه می‌شود.
۲: در هر لایه مقدار رابطه‌ی زیر محاسبه شده و به عنوان ورودی به لایه‌ی بعد داده می‌شود.

$$\mathbf{y} = f\left(\sum_{i=1}^{n} \theta_i x_i + b\right)$$

۳: مقدار رابطه‌ی زیر برای لایه‌ی خروجی محاسبه می‌شود.

$$\frac{\partial \mathbf{J}}{\partial \theta_{i,j}} = \frac{\partial \mathbf{J}}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial \theta_{i,j}}$$

برای این منظور از رابطه‌های (۱-۲۰)، (۱-۲۱) و (۱-۲-۱) استفاده می‌شود.

۴: از آخرین لایه‌ی پنهان شروع به تغییر وزن‌های لایه‌ها کرده تا به لایه‌ی ورودی برسید. برای این منظور میزان خطای نورون در هر لایه با مقدار زیر به دست می‌آید.

$$\delta_i = \left(\sum_j \delta_j \theta_{i,j}\right) \frac{\partial y_i}{\partial a_i}$$

۵: عملیات تنظیم وزن‌ها با استفاده از رابطه‌ی زیر انجام می‌شود.

$$\theta_{i,j} = \theta_{i,j} - \eta \delta_j y_i$$

۱-۳ تنسورها

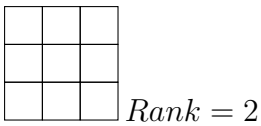
در ریاضیات به یک عدد، اسکالر گفته می‌شود. مجموعه‌ای از اسکالرها، بردار را تشکیل میدهد. مجموعه ای از بردارها در کنار هم به ماتریس تبدیل می‌شوند و مجموعهای از ماتریس‌ها را مکعب گویند. در مجموع به تمام این

موارد، تنسور میگویند. برای ذخیره اطلاعات و داده‌ها، از تنسورهای مناسب در فضای آن داده استفاده می‌شود [۴].

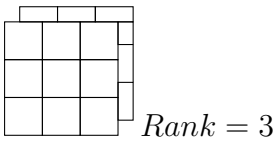
- اسکالر، تنسور صفر بعدی است که رتبه آن صفر است.

- بردار، تنسور یک بعدی با رتبه یک است.

- ماتریس، تنسور دو بعدی با رتبه دو است.



- نمونه یک تنسور سه بعدی با رتبه سه (همانند تصاویر ۳ × ۲۸ × ۲۸).

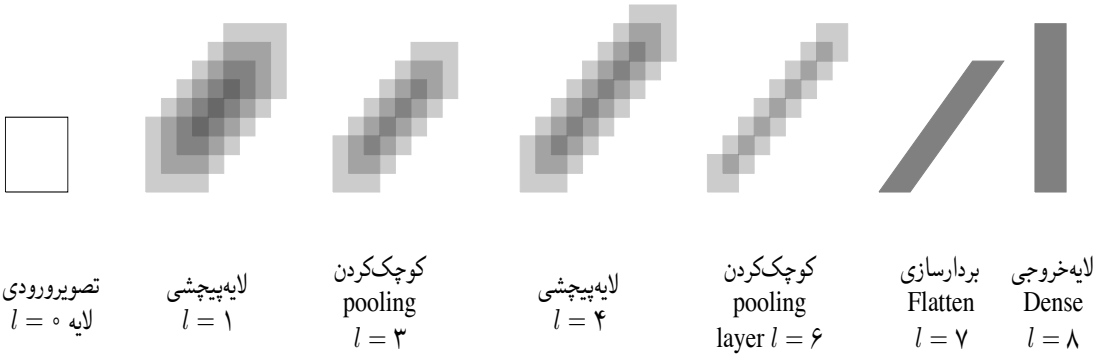


۱-۴ شبکه‌های پیچشی

برای یادگیری هزاران شیء از میلیون‌ها تصویر، به مدلی با ظرفیت یادگیری بالا و دانش قبلی نیاز است. پیچیدگی این کار به دلیل تعداد زیاد داده‌ها با ابعاد بالا، بسیار زیاد بوده و داشتن دانش اولیه در این مدل، کمک شایانی می‌کند. شبکه پیچشی [۱]، یکی از مدل‌هایی است که در این زمینه می‌توان استفاده نمود زیرا ظرفیت این شبکه‌ها با تغییر لایه، عمق و تعداد نورون، قابل کنترل است [۱۵]. با افزایش عمق،عملیات غیرخطی در لایه‌ها زیاد شده و شبکه عملکرد بهتری خواهد داشت. انواع مختلفی از معماری شبکه‌های پیچشی وجود دارد. با این حال ساختار و اجزای اصلی آن‌ها مشابه یکدیگر است. شکل ۱-۶ ساختار سلسله‌مراتبی نوعی از این شبکه را نشان می‌دهد [۱۶].

ورودی شبکه عصبی، یک بردار یک بعدی است و برای ورودی با ابعاد بالاتر، همانند یک تصویر، باید قبل از شبکه عصبی یا هر طبقه بند دیگر، از شبکه پیچشی استفاده نمود. این شبکه دارای لایه‌های متصل به هم با عمق متفاوت است. ورودی شبکه پیچشی تنسور با ابعاد مختلف مانند تصویر است. ورودی پس از عبور از چندین

[۱] Convolution Network (convnet , CNN)

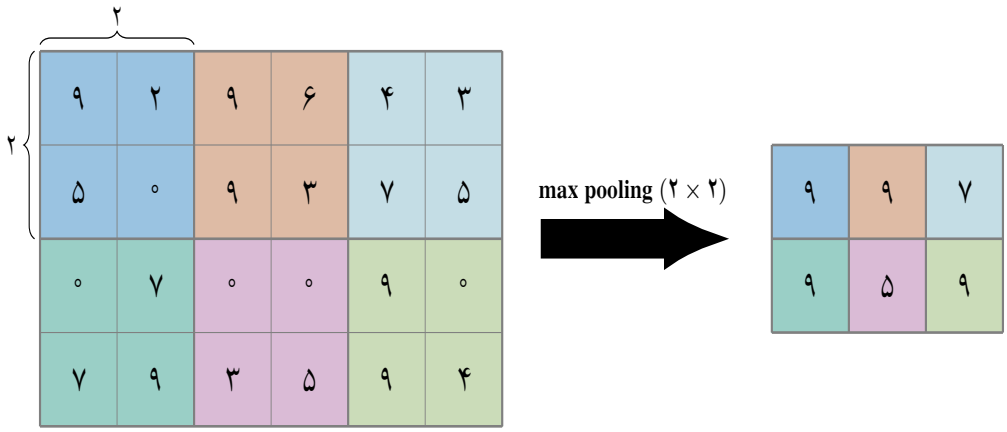


شکل ۱-۶: ساختار شبکه عصبی پیچشی

لایه، به یک بردار یک بعدی تبدیل می‌شود. خروجی لایه پیچشی به عنوان ورودی شبکه عصبی خواهد بود. لایه‌های متفاوت موجود در شبکه پیچشی، ابعاد تنسور ورودی را، بدون از دست رفتن اطلاعات مهم تصویر، تغییر می‌دهند. در این لایه‌ها از پنجره‌های لغزان (sliding window) متفاوتی استفاده می‌شود که در تمام مناطق داده‌ی ورودی اعمال می‌شود. با حرکت پنجره‌ها روی تنسور ورودی، خروجی آن قسمت را محاسبه می‌کنند. در ادامه به تعدادی از لایه‌های مورد استفاده در شبکه پیچشی، اشاره می‌شود [۴، ۱].

۱-۴-۱ کوچک کردن (pooling)

یکی از لایه‌های شبکه پیچشی که ابعاد تنسور ورودی به آن لایه را نصف می‌کند، کوچک کردن (pooling) نام دارد. این لایه با پنجره لغزان، عملیات ماکسیمم گیری max pooling و میانگین‌گیری average pooling روی ماتریس ورودی انجام می‌دهد. پنجره لغزان در هر گام حرکت با گام‌های دیگر خود هم‌پوشانی ندارد. در شکل ۱-۷ پنجره لغزان ۲ × ۲ روی ماتریس ورودی ۶ × ۴ به صورتی قرار گرفته که هم‌پوشانی نداشته باشند. سپس بیشترین مقدار عددی از هر قسمت ماتریس، که پنجره لغزان روی آن قرار گرفته را در یک بلوک از ماتریس خروجی ذخیره می‌کند، که با همان رنگ ماتریس اولیه نمایش داده شده است. ماتریس نهایی به ۳ × ۲ تبدیل شده است. این عملیات ماکسیمم گیری است. اگر در پنجره لغزان عملیات میانگین‌گیری صورت گیرد، به آن average pooling گویند [۴، ۱۶].



شکل ۱-۷: پنجره max pooling

۱-۴-۲ پیچش (Convolution)

لایه‌ی pooling فقط عملیات ریاضی ماکسیمم‌گیری یا میانگین‌گیری را انجام می‌دهد، در صورتی که شبکه پیچشی نیاز به لایه‌ای دارد که عملیات تبدیل خطی انجام دهد. لایه‌ی پیچشی (Convolution) با انجام این عملیات پایه شبکه پیچشی نامیده می‌شود [۱۶]. در این لایه تعدادی پنجره لغزان با ابعاد مختلف وجود دارد، که با گام حرکت [۱] قابل تغییر، روی ماتریس داده‌ها حرکت می‌کند. پنجره لغزان از مجموعه‌ی وزن‌های قابل یادگیری تشکیل شده است، که پارامترهای شبکه را تشکیل می‌دهند. هدف اعمال این پنجره لغزان برجسته کردن یک ویژگی خاص در یک لایه است. برای مثال در شکل ۱-۸ پنجره ۳ × ۳ روی ماتریس ۷ × ۷ با گام حرکت ۱ اعمال شده است. در یک گام حرکت پنجره، مجموع ضرایب وزن در اعداد ماتریس، به دست آمده و در یک سلول از ماتریس جواب ذخیره می‌شود [۴، ۱].

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{pmatrix}$$

شکل ۱-۸: اعمال فیلتر ۳ × ۳ روی ماتریس تصویر با گام حرکت ۱

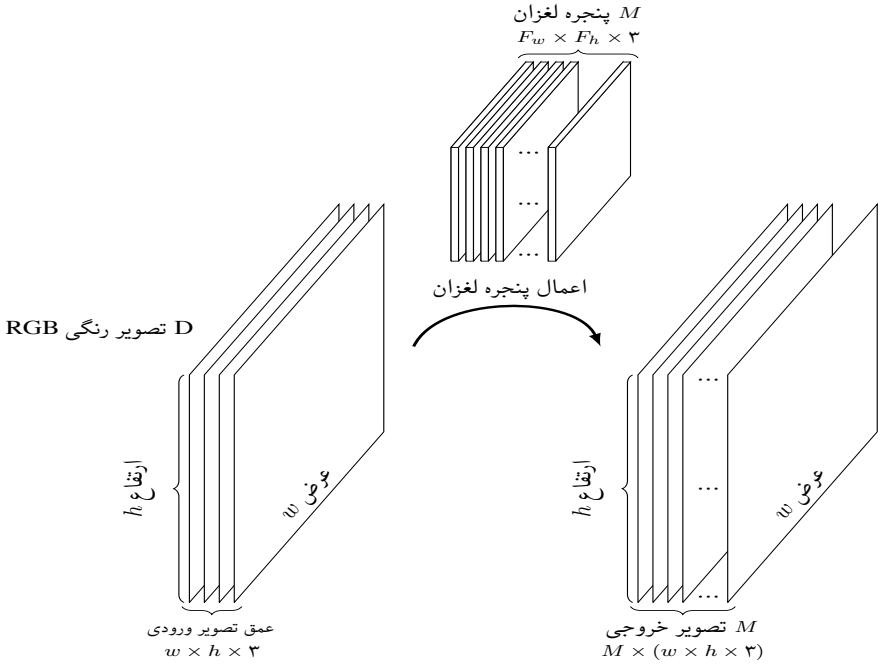
در شبکه پیچشی، داده‌ها با ابعاد بالا به صورت تنسور وارد لایه‌های این شبکه می‌شوند. در شکل ۱-۹ ورودی

[۱] stride

به صورت تنسور سه بعدی است. پنجره لغزان نیز باید تنسور باشد. بدین صورت که تعداد n پنجره‌ی لغزان بر روی نمونه‌ی ورودی اعمال می‌شود. هر پنجره یک قطعه خروجی منحصر به فرد دارد و در نهایت n تنسور سه بعدی به عنوان خروجی خواهیم داشت. طول و عرض پنجره لغزان را به ترتیب F_h و F_w در نظر بگیرید. اگر تعداد ورودی‌ها D ، تعداد پنجره‌های لغزان M و بردار بایاس $\mathbf{b}$ فرض شود، سپس تعداد کل وزن‌های θ در آن لایه‌ی پیچشی با معادله زیر محاسبه می‌شود [۱، ۴].

$$\theta = (D \times M \times F_w \times F_h) + \mathbf{b} \qquad (۲۸-۱)$$

به عنوان مثال در یک لایه پیچشی از شبکه عصبی تعداد ۶۴ پنجره‌ی لغزان ۳ × ۳ روی ۳۲ تصویر رنگی اعمال شود، تعداد وزن های لایه پیچشی برابر با $(۳۲ \times ۶۴ \times ۳ \times ۳) + ۶۴ = ۱۸۴۹۶$ است.

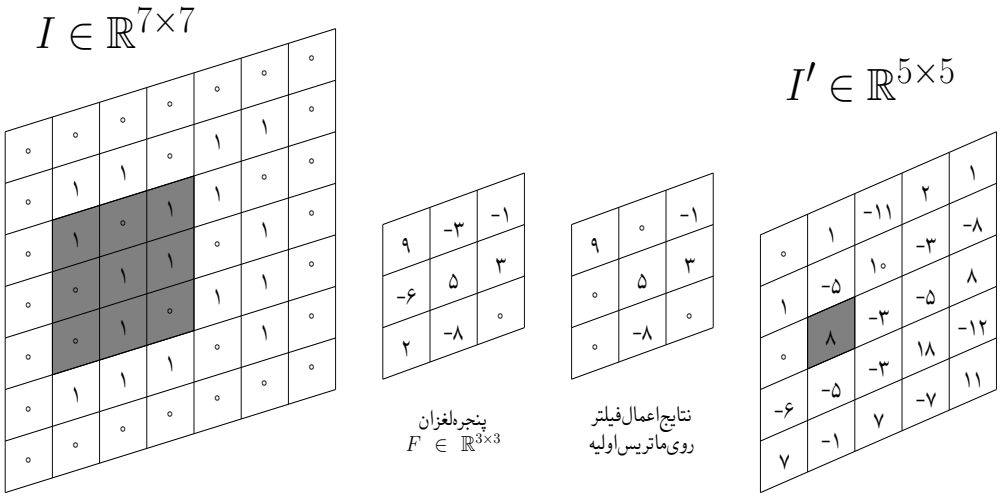


شکل ۱-۹: پنجره لغزان روی تنسورها

۱-۲-۴-۱ استفاده از padding در لایه‌ی پیچشی

مشکل استفاده از لایه‌ی پیچشی، کم کردن ابعاد ماتریس ورودی است. در شکل ۱-۸ ماتریس حاصل ۵ × ۵ است و یک سطر و یک ستون از هر طرف، نسبت به ماتریس اولیه کمتر دارد. برای جلوگیری از این مشکل می‌توان از paddingها استفاده کرد. اگر یک سطر و ستون به ابتدا و انتهای ماتریس اولیه اضافه شود، ماتریس اولیه ۹ × ۹

خواهد شد و ماتریس خروجی ۷ × ۷ می‌شود. به عملیات اضافه کردن سطر و ستون صفر، zero padding گویند. همچنین می‌توان از mirror padding استفاده کرد. بدین صورت که به جای اضافه کردن سطر و ستون صفر، سطر و ستون ابتدا و انتهای خود ماتریس را کپی کرده و به ابتدا و انتهای آن، اضافه کنیم [۴، ۱۶]. در شکل ۱-۱۰ از zero padding استفاده شده است. در شکل، ماتریس اولیه ۵ × ۵ بوده و با اضافه کردن سطر و ستون صفر به آن، ماتریس ۷ × ۷ می‌شود و پس از اعمال پنجره لغزان روی آن، ماتریس خروجی ۵ × ۵ است [۴].



شکل ۱-۱۰: اعمال zero padding روی ماتریس

اگر پنجره‌ی ۴ × ۱ روی بردار [۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹] با گام حرکت ۳ اعمال شود،به صورت زیر در مرحله اول، پنجره‌ی لغزان روی چهار عدد مربوط به گام اول قرار می‌گیرد. سپس در مرحله بعدی پنجره‌ی لغزان، چهار عدد مربوط به گام دوم را بررسی می‌کند و دو عدد آخر باقی می‌ماند.

$$\left[\underbrace{۱ ۲ ۳ ۴}\ ۵ ۶ ۷ ۸ ۹\right]$$

$$\left[۱ ۲ ۳\ \underbrace{۴ ۵ ۶ ۷}\ ۸ ۹\right]$$

در این صورت عملیات padding به دو روش زیر انجام می‌شود.

- same padding . دراین روش به تعداد عددی که کم است، صفر اضافه می‌شود تا پنجره‌ی لغزان بتواند آخرین اعداد را نیز دربر بگیرد. با این روش سه بار پنجره‌ی لغزان روی بردار قرار می‌گیرد.

$$\left[۱ ۲ ۳ ۴ ۵ ۶\ \underbrace{۷ ۸ ۹ ۰}\right]$$

- valid padding . در این روش از اعداد باقی مانده صرف نظر می‌شود و پنجره‌ی لغزان دو بار روی بردار قرار می‌گیرد.

۱-۴-۳ مسطح سازی (Flatten)

خروجی شبکه عصبی پیچشی به عنوان ورودی طبقه‌بند یا شبکه عصبی مورد استفاده قرار می‌گیرد. و ورودی طبقه‌بند و شبکه عصبی به صورت بردار است. از این رو در پایان شبکه عصبی پیچشی برای تبدیل یک تنسور با ابعاد بیشتر از ۱ به تنسور یک بعدی (بردار)، از لایه‌ی مسطح سازی Flatten استفاده می‌شود. در این روش سطرهای موجود در یک تنسور به ترتیب ترانهاده شده و زیر هم در یک بردار خروجی نشان داده می‌شود.

۱-۴-۴ Upsampling

یکی دیگر از لایه‌های موجود در شبکه پیچشی، لایه‌ی بدون وزن Upsampling است که ابعاد ورودی را دو برابر می‌کند. این لایه عکس عملیات لایه‌ی pooling (بخش۱-۴-۱) است؛ از این رو به آن unpoolingو opposite pooling نیز می‌گویند. این لایه با تکرار کردن ردیف‌ها و ستون‌های ورودی x همانند زیر، ابعاد وردی را افزایش می‌دهد. و خروجی y را تشکیل می‌دهد. این لایه همانند لایه pooling فاقد وزن است و تنها عملیات ساده ریاضی انجام می‌شود، به همین دلیل با لایه پیچشی همراه می‌شود [۱۷].

$$x = \begin{bmatrix} ۱ & ۲ \\ ۳ & ۴ \end{bmatrix} \qquad \rightarrow \qquad y = \begin{bmatrix} ۱ & ۱ & ۲ & ۲ \\ ۱ & ۱ & ۲ & ۲ \\ ۳ & ۳ & ۴ & ۴ \\ ۳ & ۳ & ۴ & ۴ \end{bmatrix}$$

۱-۴-۵ ConvolutionTranspose

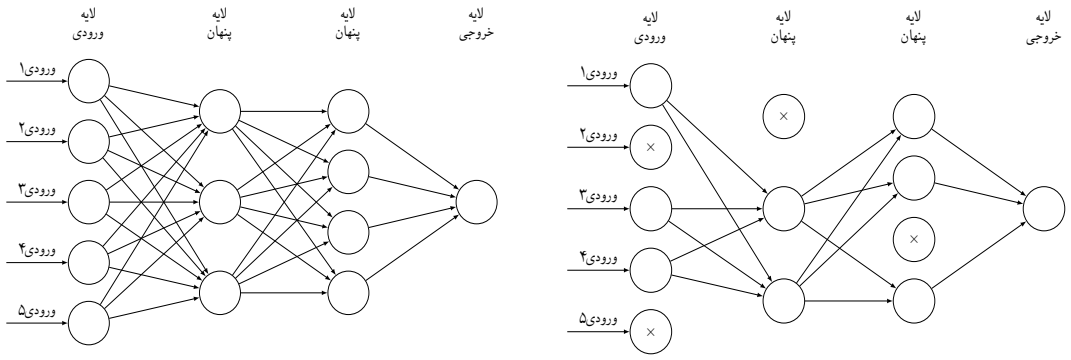
لایه‌ی ConvolutionTranspose ترکیبی از لایه‌ی پیچشی و لایه‌ی Upsampling است. در حقیقت لایه‌ی ConvolutionTranspose همانند لایه‌ی Upsampling ابعاد ورودی را دو برابر می‌کند. عملیات این لایه برعکس لایه‌ی پیچشی است از این رو به آن deconvolutional layer می‌گویند. در شکل ۱-۸ پنجره‌ی لغزان ۳ × ۳ در لایه‌ی پیچشی با گام حرکت(۱,۱) روی ماتریس ۷ × ۷ اعمال شده است و ماتریس نهایی ۵ × ۵ شده است. در لایه‌ی ConvolutionTranspose برعکس این عملیات انجام می‌شود؛ ماتریس ۵ × ۵ با اعمال

پنجره‌ی لغزان، ماتریس بزرگتر ۷ × ۷ را نتیجه می‌دهد. به عنوان مثال ماتریس ورودی x زیر را در نظر بگیرید. ابتدا سطر و ستون‌های x افزایش می‌یابد و سپس با اعمال پنجره لغزان p با گام حرکت (۲, ۲) ماتریس نهایی y حاصل می‌شود [۱۷].

$$x = \begin{bmatrix} ۱ & ۲ \\ ۳ & ۴ \end{bmatrix} \quad p = \begin{bmatrix} ۱ & ۰ \\ ۰ & ۰ \end{bmatrix} \quad \rightarrow y = \begin{bmatrix} ۱ & ۰ & ۲ & ۰ \\ ۰ & ۰ & ۰ & ۰ \\ ۳ & ۰ & ۴ & ۰ \\ ۰ & ۰ & ۰ & ۰ \end{bmatrix}$$

۵-۱ Dropout

یکی از مشکلات اساسی در شبکه‌های عصبی پیچشی، بیش‌برازش [۱] است. بدین معنی که مدل بسیار پیچیده شده است و مختص داده‌های آموزشی شده و فقط روی آن‌ها به خوبی کار می‌کند، و روی داده‌های جدید عملکرد خوبی ندارد. یکی از راه‌های جلوگیری از این مشکل حذف تصادفی برخی از نورون‌ها همانند تصویر ۱-۱۱، است. به این عمل Dropout گویند و روی خروجی هر لایه شبکه عصبی پیچشی در هنگام آموزش مدل، استفاده می‌شود. این عملیات بطور تصادفی و دوره ای برخی از نورون ها، همراه با اتصالات ورودی و خروجی آنها را از شبکه خارج می کند. Dropout از وابستگی نورون‌ها به یکدیگر جلوگیری می‌کند و همین امر باعث جلوگیری از قوی شدن مدل روی داده‌های آموزشی می‌شود. از این رو اگر مدل داده‌های جدید روبرو شود، به خوبی کار می‌کند [۴، ۱].



(آ) اعمال dropout (ب) اتصال نورون‌های کامل

شکل ۱-۱۱: حذف برخی از اتصالات نورون‌ها به صورت تصادفی

[۱] Overfit

فصل ۲

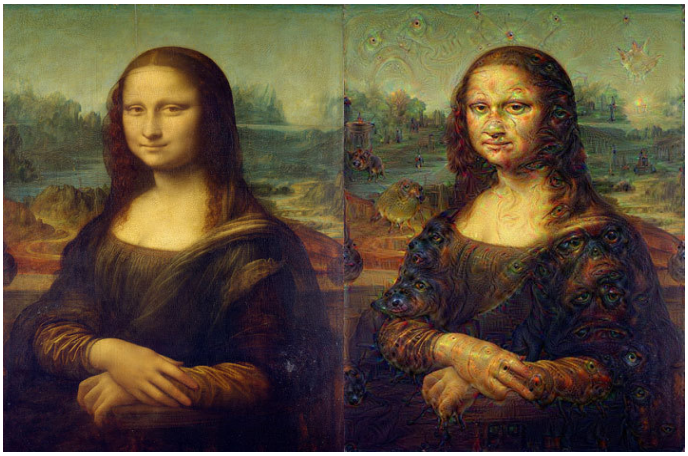
شبکه مولد رقابتی

امروزه هوش مصنوعی کاربردهای زیادی در زندگی انسان پیدا کرده است. تولید تصاویر جعلی، فیلم کوتاه و موسیقی از جمله کاربردهای نسبتاً جدید این حوزه است. تصاویر، موسیقی، ویدیو و متن‌ها دارای ساختار آماری هستند که با یادگیری فضای نهفته آماری این داده‌ها، نمونه گیری شده و آثار جدیدی ایجاد می‌شود [۴]. برای تولید این آثار از الگوریتم‌های متفاوتی استفاده می‌شود. در سال ۱۹۹۷ برای خلق آثار متنی و موسیقی از LSTM[۲] استفاده شد. شبکه‌ی عصبی بازگشتی LSTM برای ذخیره‌سازی اطلاعات و دسترسی راحت به آن‌هاست. همچنین اطلاعات را برای بازه‌های زمانی بلند مدت نگه‌داری می‌کند و بر اساس آن‌ها رفتار می‌کند. شبکه LSTM قادر به اضافه کردن اطلاعات جدید به حافظه‌ی خود و یا حذف اطلاعات از آن است. این شبکه اطلاعاتی که از قبل مشاهده کرده را با حلقه‌های بازگشتی بازیابی می‌کند و آن‌ها را به کاری که در حال انجام است، مرتبط می‌سازد. از این رو قابلیت یادگیری توالی ملودی یا متن را دارد و از آن‌ها برای خلق نمونه‌های مشابه، استفاده می‌کند [۴، ۱۸].

گوگل در سال ۲۰۱۵ برای نخستین بار اصلاح تصاویر هنری توسط رویای عمیق[۳] را انجام داد که تصاویر ورودی را با استفاده از شبکه عصبی به تصاویر جدید همراه با چشم، سگ و پرهای پرندگان مانند شکل ۲-۱ تبدیل می‌کند [۴]. تصاویر تولید شده در این زمینه همانند خواب و رویا هستند و دلیل نامگذاری همین است.

با تولید خودرمزگذارها VAE[۵] در سال ۲۰۱۳، تصاویری با حالات متفاوت از روی تصاویر موجود ساخته شد [۱۹]. شبکه عصبی مصنوعی خودرمزگزار از یک لایه‌ی ورودی، یک لایه‌ی خروجی و چند لایه‌ی پنهان ساخته شده است. ورودی خودرمزگذار، بردارهای اولیه ساختار یافته است؛ بدین منظور که این بردارها حالات خاصی از تصاویر را نمایش می‌دهند. شبکه خودرمزگزار آموزش میبیند که ورودی خود را بازسازی کند و از الگوریتم

[۲] Long short term memory [۳] DeepDream [۴] https://mastersofmedia.hum.uva.nl/blog/2015/10/30/googles-deepdream-algorithms-on-lsd/ [۵] Variational Auto-Encoders



شکل ۲-۱: تصویر تولید شده توسط رویای عمیق (سمت راست)، تصویر اصلی (سمت چپ)

پس‌انتشار خطا (۱-۱) برای یادگیری استفاده می‌کند. به عنوان مثال برای تولید چهره خندان، با اعمال بردار s که یک لبخند را تولید می‌کند، روی تنسور z (بخش ۱-۳) که تصویر یک چهره است، تصویر جدیدی به دست می‌آید که همان چهره قبلی است که لبخند می زند. در مرجع [۲۰] آموزش شبکه VAE و تصاویر تولید شده توسط این شبکه بیان شده است.

شبکه مولد رقابتی (مولد متخاصم) [۱] ، برای تولید نمونه‌های جدید مورد استفاده قرار می‌گیرد. بردار اولیه در شبکه مولد رقابتی یک فضای اولیه تصادفی (فضای پنهان) است. خروجی این شبکه نمونه‌ی جدیدی است که قبلا موجود نبوده است (مانند ساخت تصاویر جدید) [۴]. شکل ۲-۲ تصاویر تولید شده توسط شبکه مولد رقابتی را نشان می‌دهد. این شبکه روی چهره انسان‌های مشهور آموزش دیده است و تصاویری تولید کرده که متعلق به انسان‌های واقعی نیستند و توانایی شبکه‌های مولد رقابتی در تولید تصاویر با کیفیت بالا را نشان می‌دهد [۲۱]. از شبکه مولد رقابتی در حوزه مُد نیز استفاده می‌شود. در این زمینه شبکه با تجزیه و تحلیل لباس‌های مختلف، لباس با طرح جدید تولید می‌کند. در تحقیقات پزشکی از شبکه مولد رقابتی برای افزایش نمونه‌های موجود و بهبود دقت تشخیص انواع بیماری استفاده می‌شود. بدین معنا که شبکه مولد رقابتی با آموزش روی نمونه‌های موجود در این حوزه، نمونه‌های جدید تولید می‌کند و با افزایش مجوعه نمونه‌ها دقت تشخیص بیماری افزایش می‌یابد. از دیگر دستاوردهای شبکه مولد رقابتی تبدیل تصویر به تصویر دیگر [۲] و ایجاد فیلم‌های جعلی است. به عنوان مثال تبدیل تصویر سیب به تصویر پرتقال یا بالعکس و ساخت فیلم کارتونی موش و گربه. تصاویر و برنامه‌های نمونه‌های بیان شده در مرجع [۲۲] ذکر شده‌است. به طور کلی شبکه مولد رقابتی به عنوان یک ابزار مهم در هوش مصنوعی است که قادر به مطابقت با ظرفیت شناخت انسان برای به دست آوردن تخصص در تقریباً هر حوزه از مهارت‌های حرکتی، زبانی و مهارت‌های خلاقانه مانند سرودن یک غزل دیده می‌شود [۲۲]. در این

[۱] Generative Adversaial Network(GAN) [۲] image-toimage translation

فصل ابتدا به بررسی مطالب مورد نیاز به شبکه‌های مولد رقابتی می‌پردازیم. در ادامه ساختار شبکه‌های مولد رقابتی (متخاصم) و نحوه تولید تصاویر جدید را بررسی خواهیم کرد.

شکل ۲-۲: تصاویر تولید شده توسط شبکه مولد رقابتی

۲-۱ نظریه بازی

بررسی مدل‌های ریاضی و منطقی در ارتباطات اجتماعی را نظریه بازی [۱] می‌نامند. به عبارت دیگر نظریه بازی در تصمیم‌های افراد در شرایط گوناگون، کمک می‌کند. در این نظریه، همکاری یا رقابت بین دو یا چند نفر به گونه‌ای بیان می‌شود که تصمیم هر فرد به عنوان بازیکن، وابسته به تصمیم بازیکن‌های دیگر خواهد بود، و همچنین تصمیم یک بازیکن و تغییر استراتژی آن بر تصمیم بازیکن‌های دیگر تأثیر می‌گذارد. استراتژی مجموعه‌ای از رفتارها و اعمالی است که یک بازیکن برطبق شرایطی که در آن قرار می‌گیرد، انجام می‌دهد. نظریه بازی برای ارزیابی تصمیم‌گیری در شرایطی که بیش از یک نفر، موجود است، مورد استفاده قرار می‌گیرد. نظریه بازی، نخستین بار توسط جان فون نویمن [۲] معرفی شد و دارای دو شاخه‌ی زیر است [۲۳، ۲۴]:

- تعاملی (همکاری) : در این زمینه تمام بازیکن‌ها برای یک هدف تلاش می‌کنند، از این رو با یکدیگر همکاری می‌کنند. در نظریه بازی تعاملی به میزان تلاش هر بازیکن، منفعتی از نتیجه کلی به آن بازیکن تعلق می‌گیرد. به عنوان مثال برای تصمیم چند دوست برای تقسیم پول غذای سفارش داده شده، می‌توان از نظریه بازی تعاملی استفاده کرد به صورتی که هر شخص به میزان غذایی که سفارش داده است، پول پرداخت کند. در این نظریه عدالت مورد توجه قرار می‌گیرد.

---

[۱] Game Theory [۲] John von Neumann

- غیر همکارانه (رقابتی) : در این شرایط، بازیکن‌ها در رقابت با یکدیگر هستند و بهترین عملکرد موجود را انتخاب می‌کنند. این عملکرد به عملکردهای دیگر بازیکن‌ها وابسته است زیرا بر اساس تصمیمات همه‌ی بازیکن‌ها، شرایطی که در آن هستند، ایجاد شده است. از این رو هر بازیکن با توجه به تجربه‌ای که از استراتژی بازیکنان به دست آورده، بهترین تصمیم را اتخاذ می‌کند. در نظریه بازی غیرتعاملی هر بازیکن درصدد افزایش منفعت خود و همچنین افزایش زیان دیگر بازیکن‌هاست. به عنوان مثال در بازی شطرنج، هر بازیکن تصمیم‌هایی مدنظر قرار می‌دهد که رقیب را شکست دهد.

در این پایان نامه از نظریه بازی غیر همکارانه یا رقابت بین دو شبکه عصبی استفاده می‌شود. از این رو در ادامه به مسائل مربوطه در این زمینه خواهیم پرداخت.

۲-۱-۱ بازی مجموع-صفر بین دو بازیکن

بخش عمده نظریه بازی رقابتی بین دو نفر را بازی مجموع-صفر [۱] تشکیل می‌دهد. در این بازی بُرد بازیکن اول زمانی به وجود می‌آید که باخت بازیکن دوم به دست آمده باشد. به عبارتی سود یک بازیکن در ضرر بازیکن دوم است. اگر بازیکن اول را x و بازیکن دوم را y در نظر بگیریم، مقدار $u(x)$ بُرد بازیکن x را نشان می‌دهد که طبق رابطه زیر با باخت بازیکن y برابر است [۲۳].

$$u(x) = -u(y) \tag{۱-۲}$$

در این صورت مجموع بُرد دو بازیکن طبق رابطه زیر صفر است. به همین دلیل بازی مجموع صفر نامیده می‌شود.

$$u(x) + u(y) = ۰ \tag{۲-۲}$$

۲-۱-۲ کمینه-بیشینه

کمینه-بیشینه [۲] یک قانون تصمیم‌گیری برای به حداقل رساندن بیشترین ضرر یک بازیکن در بازی مجموع-صفر است. به عبارت دیگر هر بازیکن بهترین تصمیم (استراتژی) حریف را نسبت به مجموعه اقدامات (استراتژی) خود در نظر می‌گیرد و تصمیمی را اتخاذ می‌کند که بیشترین ضرر خود را کمینه کند. به عنوان مثال اگر بازیکن اول استراتژی x و بازیکن دوم استراتژی y را انتخاب کند، رابطه کمینه-بیشینه بین این دو بازیکن به صورت زیر

[۱] Zero-Sum [۲] Minimax

خواهد بود:

$$\min_{y} \max_{x} u(x, y)$$

بیشینه-کمینه [۱] یک قانون تصمیم‌گیری برای به حداکثر رساندن کمترین سود یک بازیکن در بازی مجموع-صفر است. رابطه بیشینه-کمینه از رابطه زیر محاسبه می‌شود:

$$\max_{x} \min_{y} u(x, y)$$

با برابر بودن این دو رابطه نقطه‌ی پایدار در بازی به‌وجود می‌آید که در بخش بعد مورد بررسی قرار گرفته و در قضیه ۲-۱-۱ بیان شده است. در این پایان نامه از قانون تصمیم‌گیری کمینه-بیشینه بین دو بازیکن در یک بازی مجموع-صفر استفاده می‌شود.

۲-۱-۳ تعادل نش

در بازی‌های مجموع-صفر از سری بازی‌های رقابتی که راه‌حل کمینه-بیشینه برای آن‌ها مطرح می‌شود، یک نقطه‌ی تعادل به عنوان جواب مسئله در نظر گرفته می‌شود به گونه‌ای که هر بازیکن بیشترین سود ممکن را به دست آورد. این نقطه تعادل به نام اقتصاددان و ریاضیدان آمریکایی جان فوربز نش [۲] نام‌گذاری شده و به تعادل نش [۳] معروف است. در بازی‌های مجموع-صفر، هر بازیکن باید استراتژی را انتخاب کند که سود خود را به حداکثر برساند. اگر هر دو بازیکن تصمیماتی را انتخاب کنند که حداکثر سود و حداقل ضرر را برای خود ایجاد کند، در یک تعادل نش قرار می‌گیرند [۲۳]. به عبارت دیگر تعادل نش حالتی است که هر بازیکن بهترین تصمیم (استراتژی) خود را نسبت به تصمیم بازیکن‌های دیگر انجام داده باشد. بازیکنان در تعادل نش نمی‌توانند استراتژی خود را تغییر دهند و سود بیشتری را دریافت کنند، مگر آنکه بازیکن حریف استراتژی خودش را تغییر دهد. در تعادل نش هیچ یک از بازیکن‌ها با تغییر استراتژی خود، منفعتی نخواهند داشت و از طرفی بازیکن رقیب و استراتژی‌های آن را می‌شناسد، از این رو به استراتژی که خود انتخاب کرده، پایبند می‌ماند [۲۵]. قضیه زیر بررسی نقطه تعادل در یک بازی مجموع-صفر بین دو بازیکن را بیان می‌کند.

قضیه ۲-۱-۱. (σ_1, σ_2) نقطه تعادل در یک بازی دو نفره‌ی مجموع-صفر است اگر و تنها اگر *[۲۳]*

$$\sigma_1 \in \max_{x} \min_{y} u(x, y)\ ,$$

[۱] Maximin [۲] John Forbes Nash [۳] Nash Equilibrium

$$\sigma_۲ \in \min_y \max_x u(x, y)$$

بنابراین اگر $(\sigma_۱, \sigma_۲)$ نقطه تعادل این بازی باشد، رابطه زیر را خواهیم داشت :

$$(\sigma_۱, \sigma_۲) = \max_x \min_y u(x, y) = \min_y \max_x u(x, y) \qquad (۳-۲)$$

۲-۲ شبکه مولد رقابتی

شبکه‌های مولد رقابتی نوعی سیستم یادگیر برای تولید نمونه‌های جدید از داده‌های آموزشی هستند. شبکه‌مولد رقابتی با استفاده از روش‌های یادگیری ماشین و همچنین یادگیری عمیق، مانند شبکه‌های عصبی پیچشی (بخش ۱-۴)، به تولید مدل جدید می‌پردازد. این شبکه در سال ۲۰۱۴ توسط یان گودفلو [۱] و مهدی میرزا و دیگر همکارانش ارائه شد [۲۶]. شبکه مولد رقابتی چارچوبی برای تولید مدل‌هایی است که منجر به ساخت نمونه‌های جدید می‌شود که از نمونه‌های موجود قابل تشخیص نباشند [۴].

هدف این شبکه مولد رقابتی تولید نمونه‌های جعلی همانند تصاویر، ویدیو و متن با کیفیت دنیای واقعی است. نمونه‌هایی که شبکه مولد رقابتی تولید می‌کند بستگی به انتخاب مجموعه آموزش دارد. به عنوان مثال برای تولید تصاویر از چهره انسان، شبکه مولد رقابتی روی مجموعه نمونه‌های چهره انسان آموزش می‌بیند [۲۲]. در این پایان نامه بیشتر به مبحث شبکه‌های مولد رقابتی روی تصاویر می‌پردازیم. برای ساخت نمونه‌های جدید از یک توزیع ساده نویز تصادفی شروع کرده و با استفاده از شبکه عصبی، ورودی به توزیع پیچیده مورد نظر تبدیل می‌شود.

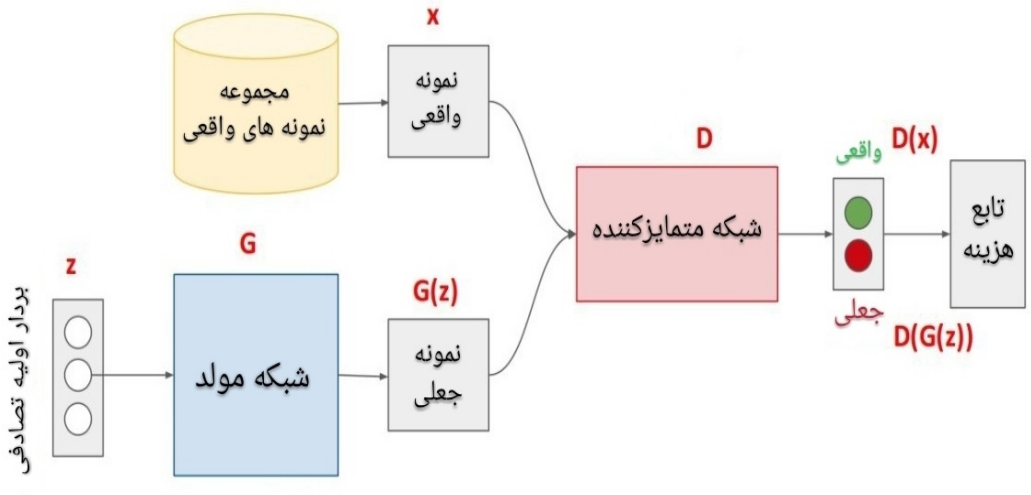
یک راه شهودی برای درک این شبکه ها، تصور جاعلی است که سعی در تولید اسکناس تقلبی دارد. در ابتدا جاعل هیچ اطلاعاتی از خصوصیات ظاهری اسکناس واقعی ندارد و نمونه اولیه را تولید می‌کند که شباهتی با اسکناس واقعی نخواهد داشت. پلیس برای هر اسکناس ارزیابی صحیح انجام می‌دهد و جعلی بودن نمونه‌ی اولیه را بازگو می‌کند. جاعل برای بار دوم، سعی در ساخت مجدد آن می‌کند. پلیس که با اسکناس واقعی آشنایی کامل دارد، مجدداً جعلی بودن نمونه را تشخیص می‌دهد. با گذشت زمان جاعل در تقلید از اسکناس واقعی، مهارت پیدا می‌کند و پلیس در زمینه یافتن اسکناس جعلی، متخصص می‌شود. در پایان این رقابت، اسکناس جعلی ساخته شده است که شباهت زیادی به اسکناس واقعی داشته به گونه‌ای که پلیس متوجه جعلی بودن آن نمی‌شود [۲۶].

---

[۱] Ian Goodfellow

۲-۳ ساختار شبکه مولد رقابتی

در شکل ۲-۳ ساختار یک شبکه مولد رقابتی نشان داده شده است [۱]. طبق شکل، دو شبکه عصبی (۱-۱) استفاده شده در شبکه مولد رقابتی، شبکه مولد و شبکه متمایز کننده نام دارد. شبکه مولد رقابتی از رقابت بین این دو شبکه عصبی، در یک نظریه بازی (بخش ۲-۱)، به منظور تولید نمونه‌های جدید، استفاده می‌کند.

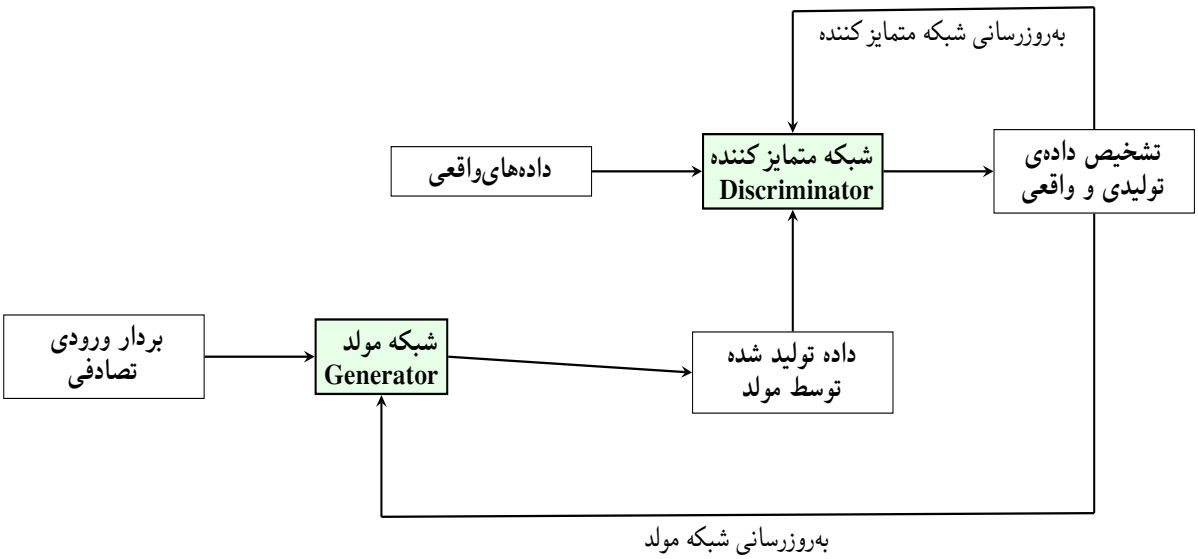


شکل ۲-۳: ساختار شبکه مولد رقابتی

هر یک از شبکه‌های مولد و متمایزکننده در پی بُرد خود و شکست رقیب است. بدین صورت که در ابتدا بردار اولیه تصادفی، به شبکه مولد داده می‌شود و این شبکه نمونه‌ی جدیدی به ابعاد تصاویر مورد نظر تولید می‌کند. نمونه‌های واقعی با برچسب ۱ و نمونه‌های تولید شده با برچسب صفر به شبکه متمایزکننده داده می‌شود. این شبکه تفاوت میان نمونه‌های واقعی و نمونه‌های تولید شده را آموزش میبیند. سپس مجدداً نمونه‌های جدید توسط مولد تولید شده و همراه داده‌های اصلی با برچسب ۱، به شبکه متمایزکننده داده می‌شود. شبکه متمایزکننده یاد گرفته که چه داده‌ای واقعی است، از این رو نمونه‌های ساخته شده توسط مولد را شناسایی می‌کند و در خروجی برچسب صفر به آن‌ها می‌دهد. در این حالت، هر دو مدل همانند شکل ۲-۴ با استفاده از الگوریتم های پس انتشار خطا (بخش ۱-۲) به‌روزرسانی می‌شوند. فرآیند یادگیری تا زمانی که سیستم در تعادل نش [۲] قرار گیرد ادامه می‌یابد. با تکرار این روند شبکه مولد می‌آموزد که نمونه‌هایی با شباهت بیشتر به نمونه‌های اصلی، تولید کند و شبکه متمایزکننده را فریب دهد. در مقابل شبکه متمایزکننده می‌آموزد تا داده‌های ساخته شده را از نمونه‌های واقعی، جدا کند و رقیب را شکست دهد. با تکرار این روند در هر بار اجرا، شباهت نمونه‌ی تولید شده توسط مولد، به نمونه‌ی اصلی، بیشتر می‌شود و همچنین نمونه‌های جعلی با دقت بیشتری، شناسایی می‌شوند [۲۶، ۴، ۱].

[۱] https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf
[۲] Nash equilibrium

در ادامه ساختار شبکه مولد و شبکه متمایزکننده، مورد بررسی قرار می‌گیرد و سپس آموزش یک شبکه مولد رقابتی را بیان خواهیم کرد.



شکل ۲-۴: ساختار شبکه مولد رقابتی و به روز رسانی وزن‌های شبکه‌ها با قاعده پس انتشار خطا

۲-۳-۱ شبکه متمایزکننده

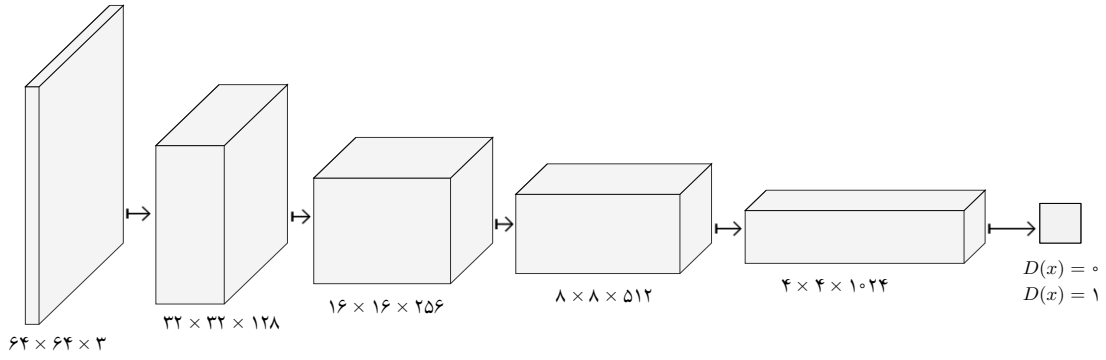
یکی از موفقیت ها در یادگیری عمیق، شبکه‌های متمایزکننده[1] هستند. این شبکه آموزش می‌بیند تا نمونه های توزیع داده‌های موجود و توزیع مدل مولد را از یکدیگر تشخیص دهد. شبکه متمایز کننده نمونه‌های اصلی و نمونه‌های ساخته شده توسط مولد را به عنوان ورودی می‌گیرد و در خروجی مشخص می‌کند کدام ورودی جزء داده‌های واقعی و کدام ورودی، ساخته شده توسط مولد است. شبکه متمایزکننده به نوعی یک طبقه‌بند محسوب می‌شود، زیرا نمونه‌های تولیدی توسط شبکه مولد و نمونه‌های واقعی را طبقه‌بندی می‌کند [۲۷، ۲۶].

طبق شکل ۲-۵ متمایز کننده یک شبکه عصبی است که از لایه‌های متفاوت تشکیل شده است (در فصل اول با شبکه عصبی پیچشی آشنا شدیم). ورودی این شبکه عصبی، تنسور با ابعاد مختلف است. به عنوان مثال تصاویر ۲۸ × ۲۸ ورودی این شبکه خواهند بود، و خروجی اعداد باینری ۰ یا ۱ خواهد بود. در صورتی که خروجی احتمالاتی نزدیک به ۱ باشد، تصویر از تصاویر اصلی داده های آموزشی است، در غیر این صورت اگر نزدیک به صفر بود، تصویر تولید شده توسط مولد است [۴].

نوع و تعداد لایه‌های مورد استفاده در این شبکه، در معماری‌های مختلف متفاوت است. عموماً از لایه‌هایی استفاده می‌شود که ابعاد ورودی را کاهش دهد، و لایه‌ی پیچشی که در بخش ۱-۴ مورد بررسی قرار گرفته است

[1] Discriminator

[۲۷، ۴]. به عنوان مثال یک مدل شبکه متمایزکننده در قطعه برنامه آ-۲ موجود است [۲۸]. ورودی این مدل، تصویر ۳ × ۲۸ × ۲۸ است. مدل دارای دو لایه‌ی پیچشی Conv2D با ۶۴ پنجره‌ی لغزان (۳, ۳) با گام حرکت (۲, ۲) است که تصویر را به ۶۴ × ۱۴ × ۱۴ و سپس به ۶۴ × ۷ × ۷ تبدیل می‌کنند. در این لایه‌ها از same padding (بخش ۱-۴-۲-۱) استفاده شده است. تابع فعالیت مورد استفاده نیز Leaky ReLU با شیب ۰٫۲ است. از لایه‌ی مسطح سازی (بخش ۱-۴-۳) برای تبدیل تنسور چند بعدی ورودی به بردار یک بعدی استفاده می‌شود. در لایه‌ی آخر تابع فعالیت سیگموئید (جدول ۱-۱) قرار دارد زیرا ورودی را به اعداد باینری تبدیل کند. بدین صورت طبقه‌بندی در مدل متمایزکننده انجام می‌شود.



شکل ۲-۵: ساختار شبکه عصبی متمایزکننده

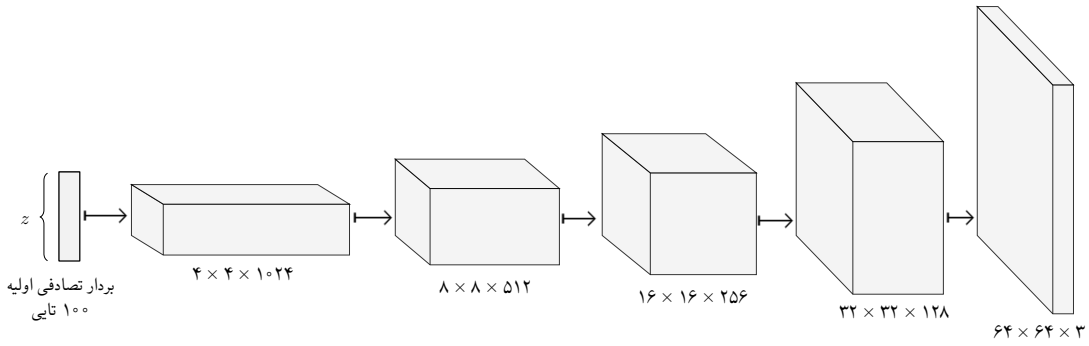
۲-۳-۲ شبکه مولد

شبکه مولد [۱] یک شبکه عصبی با لایه های متفاوت همانند شکل ۲-۶ است که در فصل قبل با شبکه عصبی آشنا شدیم. شبکه مولد به‌جای استخراج ویژگی، ویژگی‌های خاص را در کنار یکدیگر قرار می‌دهد تا نمونه‌ی نهایی را تولید کند از این رو برای تولید توزیع های پیچیده (مانند تصاویر)، از این شبکه استفاده می شود. شبکه مولد یک بردار نویز تصادفی (مثل بردار نویز ۱۰۰تایی) را به عنوان ورودی دریافت می‌کند، و با افزایش ابعاد آن توسط لایه‌های مختلف شبکه، نمونه‌ی جدید با ابعاد متناسب را تولید می‌کند [۲۲]. برای این منظور مولد از لایه‌های مختلفی استفاده می‌کند.

در ساختار شبکه عموماً از لایه‌هایی استفاده می‌شود که ابعاد تنسور ورودی را به ابعاد خروجی مورد انتظار، تبدیل کند [۴]. برای نمونه یک مدل شبکه مولد در قطعه برنامه آ-۳ موجود است [۲۸]. ورودی این مدل شبکه مولد بردار اولیه ۱۰۰تایی است و با استفاده از لایه‌ی Dense، تعداد آن را افزایش داده و سپس با کمک Reshape، آن را به ۶۴ تنسور ۷ × ۷ تبدیل می‌کند. دو لایه‌ی پیچشی با پنجره‌ی لغزان ۴ × ۴ و گام حرکت (۲, ۲) در این

[۱] Generative

مدل وجود دارد و در نهایت خروجی یک تصویر ۲۸ × ۲۸ با سه کانال رنگی [1] و مقادیر در محدوده [۱,۱−] خواهد بود.



شکل ۲-۶: ساختار شبکه عصبی مولد

۲-۴ آموزش شبکه مولد رقابتی

اکثر مدل های یادگیری عمیق بر روی مدل هایی با پارامترهای تابع توزیع احتمال، متمرکز شده اند. بدین صورت که تابع توزیع احتمال نمونه‌ها، موجود است و با داشتن آن، نمونه‌های جدید، تولید شده یا مورد ارزیابی قرار می‌گیرد. به عنوان مثال در طبقه‌بندی دو کلاسه، با در نظر گرفتن مقدار t به عنوان خروجی واقعی نمونه‌ها و مقدار y به عنوان خروجی پیش‌بینی شده، احتمال اینکه عنصر x متعلق به کلاس ۱ و یا ۱− باشد، با درنظر گرفتن پارامتر θ به صورت زیر خواهد بود:

$$P(t = ۱|x; \theta) = y_\theta(x)$$
$$P(t = ۰|x; \theta) = ۱ - y_\theta(x)$$

هدف پیدا کردن پارامتر بهینه θ است به گونه‌ای که مقدار پیش‌بینی شده به مقدار جواب واقعی نزدیک باشد و میزان خطا کاهش یابد. از این رو روابط بالا را ترکیب کرده و به رابطه زیر می‌رسیم:

$$P(t|x\,; \theta) = (y_\theta(x))^t\,(۱ - y_\theta(x))^{۱-t}$$

[1] RGB

این رابطه باید برای تمام داده‌های آموزشی برقرار باشد از این رو تابع به صورت ضرب توابع احتمالی بازنویسی می‌شود، که به آن تابع درست‌نمایی [1] می‌گویند:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{N} P(t_i|x_i\,;\theta) \\ &= \prod_{i=1}^{N} (y_\theta(x_i))^{t_i}\,(1-y_\theta(x_i))^{1-t_i} \end{aligned} \tag{۴-۲}$$

این مدل ها با استفاده از بیشینه کردن لگاریتم تابع درست‌نمایی به صورت زیر آموزش میبینند:

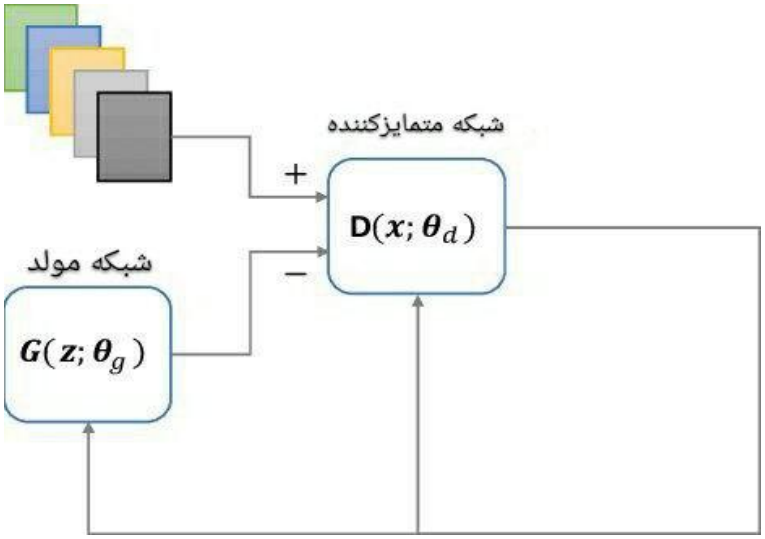
$$\log L(\theta) = \sum_{i=1}^{N} t_i \log y(x_i) + (1-t_i)\log(1-y(x_i)) \tag{۵-۲}$$

برای به دست آوردن پارامتر بهینه θ که رابطه بالا را بیشینه کند، به تقریب های بیشمار یا مشتق از تابع احتمال نیاز است. از طرفی این مدل‌ها باعث تولید نمونه‌های بیش از حد متنوع می‌شوند؛ مانند تصویر یک سگ با چند سر و یا تصویر یک زرافه با ده چشم که در دنیای واقعی وجود ندارند. با ساخت ماشین‌های مولد، پیچیدگی آموزش کاهش یافته است زیرا مدل‌های ماشین‌های مولد، به صراحت بیانگر احتمال نیستند، در عین حال قادر به تولید نمونه هایی از توزیع مطلوب هستند. بدین معنا که ماشین مولد بدون داشتن تابع توزیع نمونه‌ها، قادر به تولید نمونه‌هایی از آن‌هاست. شبکه‌های مولد رقابتی نمونه‌ای از یک دستگاه تولیدی است که می‌تواند با استفاده از روش‌های پس انتشار خطا به جای تقریب‌های بیشمار، آموزش داده شود [۲۶، ۲۲]. در شبکه‌های عصبی و شبکه‌های عصبی پیچشی، روش سریع‌ترین شیب، تپه نوردی [2] در یک محیط ایستاست. همانند شکل ۱-۴ محیط یکسان است و هیچ تغییری در آن صورت نمی‌گیرد. بدین صورت کمینه سراسری ثابت است و می‌توان با به‌هنگام سازی وزن‌ها به آن رسید. اما با یک شبکه مولد رقابتی، هر قدم که از پایین تپه برداشته می شود، کل محیط را کمی تغییر می‌دهد و یک سیستم پویاست. در سیستم پویا، فرآیند بهینه سازی به دنبال کمینه نیست بلکه به دنبال تعادل بین دو نیرو است. ایجاد یک شبکه مولد رقابتی به تنظیم مدل و پارامترهای آموزش نیاز دارد، که در ادامه به آن‌ها می‌پردازیم [۴].

شبکه مولد و شبکه متمایزکننده به صورت مجزا ساخته می‌شوند و هر کدام به صورت جداگانه آموزش می‌بینند. شبکه مولد رقابتی با اجرای متناوب این دو شبکه در خلال یک شبکه بزرگ، به ساخت تصاویر مورد نظر دست می‌یابد. همانند شکل ۲-۷ در شبکه مولد رقابتی، شبکه مولد $\mathbf{G}(z,\theta_g)$ با داشتن وزن‌های θ_g که پارامترهای شبکه مولد است، روند تولید تصویر جدید را شروع می‌کند. بدین منظور بردار اولیه تصادفی z را به عنوان ورودی می‌گیرد. بردار اولیه از تابع توزیع تصادفی نرمال [۲۹] ($z \sim N(0,1)$) و یا از تابع توزیع تصادفی یکنواخت

[1] maximum likelihood

[2] روش چک کردن تمام همسایگی‌ها و رسیدن به نقطه بهینه



شکل ۲-۷: روند تولید تصاویر جدید با شبکه مولد رقابتی

($z \sim U(-۱, ۱)$) تبعیت می‌کند، و دارای توزیع احتمال $P_z(z)$ است. x با توزیع $P_g(x)$ خروجی شبکه مولد خواهد بود. فرض کنید $\mathbf{J}(\mathbf{G})$ تابع هزینه در شبکه مولد است که در بخش ۱-۱-۲ انواع تابع هزینه بیان شده است.

نمونه‌های تولیدشده توسط مولد که از تابع توزیع $P_g(x)$ تبعیت می‌کنند ($x \sim P_g(x)$) همراه با مقادیر داده اصلی که از توزیع $P_{data}(x)$ تبعیت می‌کنند ($x \sim P_{data}(x)$)، به عنوان ورودی شبکه متمایز کننده $\mathbf{D}(x, \theta_d)$ هستند. شبکه متمایزکننده یک طبقه‌بند باینری با وزن‌های θ_d است، که خروجی احتمالاتی آن برای داده‌های واقعی، نزدیک به ۱ و برای داده‌های ساخته شده توسط مولد، نزدیک به صفر خواهد بود. تابع هزینه مربوط به شبکه متمایزکننده با $\mathbf{J}(\mathbf{D})$ نشان داده می‌شود. [۱].

در آموزش شبکه‌های مولد رقابتی برخلاف شبکه‌های عصبی، دو شبکه وجود دارد. از این رو از یک بازی متوالی [۱] بین دو شبکه استفاده می‌کنیم. بدین معنا که بازیکن‌ها (شبکه مولد و شبکه متمایزکننده) به نوبت استراتژی خود را انتخاب می‌کنند (برنامه‌ها اجرا می‌شوند). در ابتدا شبکه متمایزکننده با تنظیم وزن‌های θ_d سعی در کاهش میزان خطا و کمینه کردن تابع هزینه $\mathbf{J}(\mathbf{D})$ که در بخش ۲-۴-۱ تعریف می‌شود، دارد و سپس شبکه مولد تلاش می‌کند با تنظیم وزن‌های θ_g میزان تابع هزینه $\mathbf{J}(\mathbf{G})$ (بیان شده در بخش ۲-۴-۲) را کمینه سازد. به عبارتی در ابتدا شبکه متمایزکننده و سپس شبکه مولد آموزش می‌بینند. روند این بازی به صورت همزمان صورت نمی‌گیرد [۱].

شبکه‌های مولد رقابتی از نظریه بازی غیرتعاملی (۲-۱) بین دو شبکه مولد و متمایزکننده، استفاده می‌کند. از این رو شبکه مولد و شبکه متمایزکننده در رقابت با یکدیگر هستند. شبکه مولد با ساخت نمونه‌های جعلی سعی در فریب شبکه متمایزکننده دارد. از طرف دیگر متمایزکننده با تشخیص نمونه‌های جعلی از نمونه‌های واقعی،

[۱] Sequential

شبکه مولد را شکست می‌دهد. در این بازی سود یا زیان شبکه مولد با سود یا زیان شبکه متمایزکننده، متعادل است. بدین معنا که مجموع توابع هزینه شبکه مولد و شبکه متمایزکننده، صفر است. رابطه مجموع-صفر (۲-۱) برای توابع هزینه این شبکه‌ها به صورت زیر برقرار است [۲۲]:

$$\mathbf{J}(\mathbf{G}) = -\mathbf{J}(\mathbf{D}) \tag{۲-۶}$$

در آموزش شبکه‌های مولد رقابتی، یک بازی کمینه-بیشینه (بخش ۲-۱-۱) بین شبکه مولد و شبکه متمایزکننده برقرار است. بدین معنا که استراتژی بازیکن اول (شبکه مولد) به حداقل رساندن، حداکثر امتیاز حریف (شبکه متمایزکننده) است. به همین دلیل کمینه-بیشینه نامیده شده است. در این نوع بازی هنگامی که یک بازیکن با یک مقدار مشخص بهبود می‌یابد، بازیکن دیگر با همان مقدار بدتر می‌شود. از این رو هر بازیکن تلاش می‌کند که حریف خود را شکست دهد [۱].

شبکه متمایزکننده آموزش میبیند تا به درستی بین نمونه‌های اصلی و نمونه‌های ساخته شده توسط مولد، تمایز قائل شود و میزان تابع هزینه $\mathbf{J}(\mathbf{D})$ را کمینه کند که در بخش ۲-۴-۱ بررسی خواهد شد. از طرف دیگر شبکه مولد نیز سعی می‌کند با ساخت تصاویر جدید، از شبکه متمایزکننده پیشی بگیرد و آن را فریب دهد. پس در تلاش است که میزان تابع هزینه $\mathbf{J}(\mathbf{G})$ را کمینه کند. این معادل است با این که شبکه مولد با ساخت تصاویر جدید شبکه متمایزکننده را فریب داده به صورتی که شبکه متمایزکننده قادر به تشخیص ساختگی بودن تصویر نباشد و میزان تابع هدف متمایزکننده را افزایش دهد که در بخش ۲-۴-۲ بررسی خواهد شد. بدین صورت رابطه زیر برقرار خواهد بود.

$$\min \mathbf{J}(\mathbf{G}) \cong \max \mathbf{J}(\mathbf{D}) \tag{۲-۷}$$

پس از چند مرحله آموزش، مقدار $\mathbf{J}(\mathbf{D})$ و $\mathbf{J}(\mathbf{G})$ در کمینه محلی [۱] قرار می‌گیرند. از این رو جواب بازی کمینه-بیشینه در حالت تعادل نش (بخش ۲-۱-۳) قرار دارد. بدین معنی که یک بازیکن بدون توجه به آنچه ممکن است بازیکن دیگر انجام دهد، نمی‌توانند عملکرد خود را بهبود دهند. تعادل نش در شبکه مولد رقابتی زمانی است که شبکه مولد به حدی خوب عمل کند (نمونه‌های جعلی تولید شده، از نمونه‌های واقعی قابل تشخیص نباشد) که شبکه متمایزکننده قادر به تفکیک نمونه تولید شده، نباشد. بدین صورت که شبکه متمایزکننده در بهترین حالت به صورت تصادفی حدس می‌زند که یک نمونه واقعی یا جعلی است (۵۰ درصد حدس می‌زند نمونه واقعی است و ۵۰ درصد می‌گوید که نمونه جعلی است). به عبارتی خروجی متمایزکننده صرف نظر از ورودی، همیشه $\frac{۱}{۲}$

[۱] Local minimum

باشد [۱، ۲۲]. در این حالت نمونه‌های جعلی ساخته شده توسط مولد مشابه نمونه‌های واقعی است و شبکه متمایزکننده راهی برای تشخیص نمونه‌ها از یکدیگر ندارد. به دلیل آن که نیمی از نمونه‌هایی که این شبکه دریافت می‌کند از نمونه‌های واقعی و نیمی از نمونه‌های جعلی ساخته شده توسط مولد است، شبکه متمایزکننده هر نمونه را به صورت واقعی یا جعلی با احتمال $\frac{1}{2}$ طبقه‌بندی می‌کند. پس نمونه‌های ساخته شده توسط شبکه مولد در حالتی قرار گرفته که توسط شبکه متمایزکننده قابل تشخیص نیست، و با تغییر جزئی در فرآیند مورد استفاده برای تبدیل بردار نویز تصادفی z به یک نمونه‌ی جعلی، ممکن است به شبکه متمایزکننده نشان بدهد که چگونه این نمونه‌ها را از نمونه‌های واقعی تشخیص بدهد. به همین دلیل شبکه مولد در حالتی است که از تنظیم بیشتر، چیزی برای به دست آوردن ندارد و در تعادل باقی می‌ماند. شبکه مولد رقابتی با تعادل ایجاد شده بین شبکه مولد و شبکه متمایزکننده، به همگرایی رسیده است [۲۲].

۲-۴-۱ آموزش شبکه متمایز دهنده

شبکه متمایزکننده، یک شبکه عصبی برای طبقه بندی داده‌های واقعی و داده‌های ساخته شده توسط مدل است. آموزش شبکه متمایزکننده با توجه به نوع ورودی، به دو صورت انجام می‌گیرد [۱]:

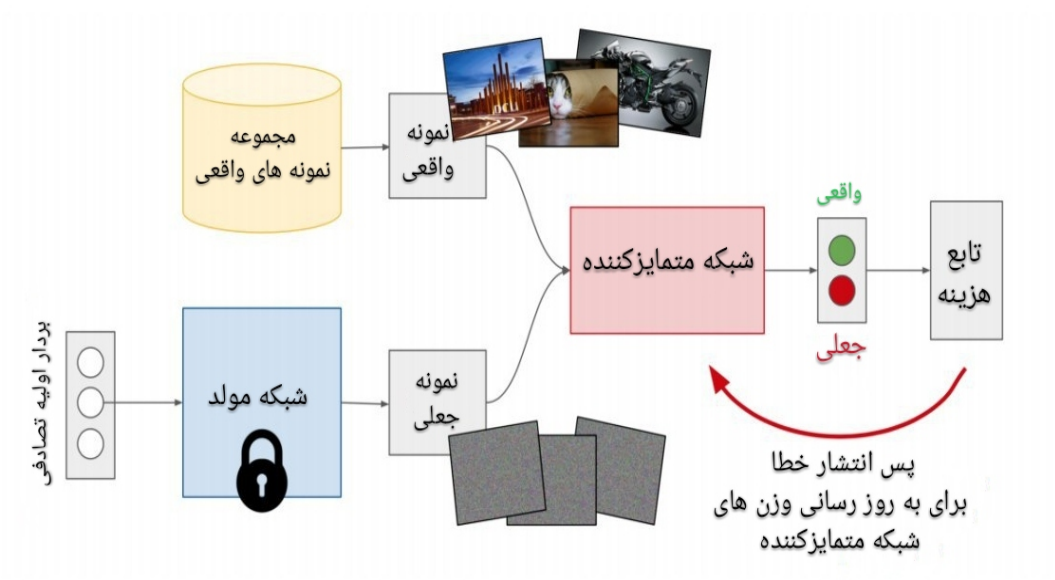
- اگر ورودی شبکه متمایزکننده از داده‌های واقعی باشد یعنی $x \sim P_{data}(x)$، خروجی این شبکه مقدار $\mathbf{D}(x)$ خواهد بود. در این صورت شبکه مولد فعالیتی نداشته و شبکه مولد رقابتی از یک شبکه متمایزکننده تشکیل شده و آن را آموزش می‌دهد.

- اگر ورودی شبکه متمایزکننده از داده‌های تولید شده توسط مولد باشد ($x \sim P_g(x)$). در این صورت شبکه مولد و شبکه متمایزکننده به عنوان یک شبکه بزرگ با هم کار می‌کنند. شبکه مولد از بردار اولیه تصادفی z شروع می‌کند و از آن برای ساخت نمونه‌ی جدید $\mathbf{G}(z)$ استفاده می‌کند. سپس این نمونه ساخته شده به عنوان ورودی به شبکه متمایزکننده داده می‌شود و این شبکه مقدار $\mathbf{D}(\mathbf{G}(z))$ را در خروجی نمایش می‌دهد.

در هر دو مرحله بالا مقدار تابع هزینه محاسبه می‌شود و میزان خطا با الگوریتم پس انتشار خطا همانند شکل ۲-۸ به شبکه متمایزکننده برمی‌گردد [۱] و وزن‌های این شبکه همانند بخش ۱-۱-۳ به‌روز می‌شود. در طول آموزش شبکه متمایزکننده، شبکه مولد قفل خواهد بود و آموزش نمیبیند. ثابت ماندن وزن‌های شبکه مولد و به‌روز شدن وزن‌های شبکه متمایزکننده، پیشرفت کارایی شبکه متمایزکننده را تضمین می‌کند [۱].

شبکه متمایزکننده یک شبکه عصبی برای طبقه‌بندی نمونه‌های واقعی و جعلی است از این رو برای آموزش خود نیاز به لگاریتم تابع درست‌نمایی (۲-۵) دارد که در ابتدای بخش ۲-۴ به آن اشاره شده است. در این رابطه

[۱] https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf



شکل ۲-۸: آموزش شبکه متمایزکننده در GAN

$y(x_i)$ احتمال جواب پیش بینی شده و t_i احتمال جواب واقعی است. رابطه بیان شده را برای یک دسته mتایی از نمونه‌ها به صورت زیر بازنویسی می‌کنیم:

$$\mathbf{J} = -\frac{۱}{m}\sum_{i=۱}^{m}\left[\, t_i \log(y(x_i)) + (۱ - t_i)\log(۱ - y(x_i))\,\right] \qquad (۸-۲)$$

با در نظر گرفتن E به عنوان مقدار مورد انتظار یک متغیر تصادفی (امید ریاضی[۱])، رابطه تابع هزینه برای شبکه متمایزکننده به صورت زیر خواهد بود:

$$\mathbf{J}(\mathbf{D}) = -\frac{۱}{۲}E_{x\sim p_{data}}\left[\log \mathbf{D}(x)\right] - \frac{۱}{۲}E_{z\sim p_z}\left[\log\left(۱ - \mathbf{D}(\mathbf{G}(z))\right)\right] \qquad (۹-۲)$$

در حالت کلی به دنبال کمینه کردن این تابع هستیم. $\mathbf{J}(\mathbf{D})$ از دو بخش با علامت منفی تشکیل شده است پس با افزایش مقادیر دو قسمت، کاهش کلی تابع را خواهیم داشت. در عبارت $\frac{۱}{۲}E_{x\sim p_{data}} \log \mathbf{D}(x)$، مقدار تابع هزینه شبکه متمایزکننده هنگامی که ورودی از داده‌های واقعی باشد را نشان می‌دهد. در این قسمت شبکه مولد هیچ نقشی ندارد و به دلیل آنکه ورودی‌ها از داده‌های واقعی هستند، خروجی شبکه متمایزکننده $\mathbf{D}(x)$ برابر یک خواهد بود. در این صورت مقدار بیشینه خود را دارد و به کمینه کردن تابع کلی کمک می‌کند. عبارت دوم یعنی $\frac{۱}{۲}E_{z\sim p_z} \log\left(۱ - \mathbf{D}(\mathbf{G}(z))\right)$، مقدار تابع هزینه شبکه متمایزکننده است وقتی که ورودی این شبکه از داده‌های تولید شده توسط مولد باشد. زمانی این مقدار بیشینه می‌شود که $\mathbf{D}(\mathbf{G}(z))$ برابر صفر شود، یا به عبارتی شبکه

[۱] Expectation Value

متمایزکننده به درستی داده‌های تولید شده را از داده‌های اصلی تفکیک کند. به طور خلاصه زمانی که برای همه‌ی $x \sim P_{data}(x)$ مقدار $\mathbf{D}(x) = ۱$ شود و همچنین برای همه‌ی $x \sim P_g(x)$ یا $x = \mathbf{G}(z)$ مقدار $\mathbf{D}(x) = ۰$ شود، این دو عبارت مقدار بیشینه خود را خواهند داشت و به دلیل وجود علامت منفی، مقدار کلی کمینه می‌شود [۱].

۲-۴-۱-۱ الگوریتم شبکه عصبی متمایزکننده

چارچوب کلی آموزش شبکه متمایزکننده در الگوریتم ۲-۱ آمده است. نمونه‌های واقعی با برچسب یک و نمونه‌های ساخته شده با برچسب صفر به شبکه متمایزکننده داده می‌شود. شبکه متمایزکننده تحت آموزش قرار می‌گیرد تا نمونه‌های جعلی و نمونه‌های واقعی را متمایز کرده و طبقه‌بندی کند. به عنوان مثال برنامه پایتون آ-۴ موجود در پیوست، آموزش شبکه متمایزکننده را نشان می‌دهد [۳۰].

الگوریتم ۲-۱ الگوریتم آموزش شبکه متمایزکننده

ورودی: نمونه‌های ساخته شده توسط مولد $x \sim P_g(x)$
و نمونه‌های آموزشی $x \sim P_{data}(x)$
خروجی: طبقه‌بندی نمونه‌های واقعی و جعلی

۱: یک دسته‌ی mتایی از نمونه‌های آموزشی به صورت تصادفی انتخاب می‌شود و به همراه برچسب ۱به شبکه داده می‌شود.

$$\left[\begin{array}{cccc} x_۱ & x_۲ & x_۳ & \dots & x_m \end{array} \right] \sim P_{data}(x)$$

وزن‌های شبکه برای کمینه سازی تابع هزینه زیر توسط الگوریتم پس انتشار خطا (۱-۱) به‌هنگام می‌شوند.

$$\mathbf{J}(\mathbf{D}) = -\frac{۱}{۲} E_{x \sim p_{data}} \left[\log \mathbf{D}(x) \right]$$

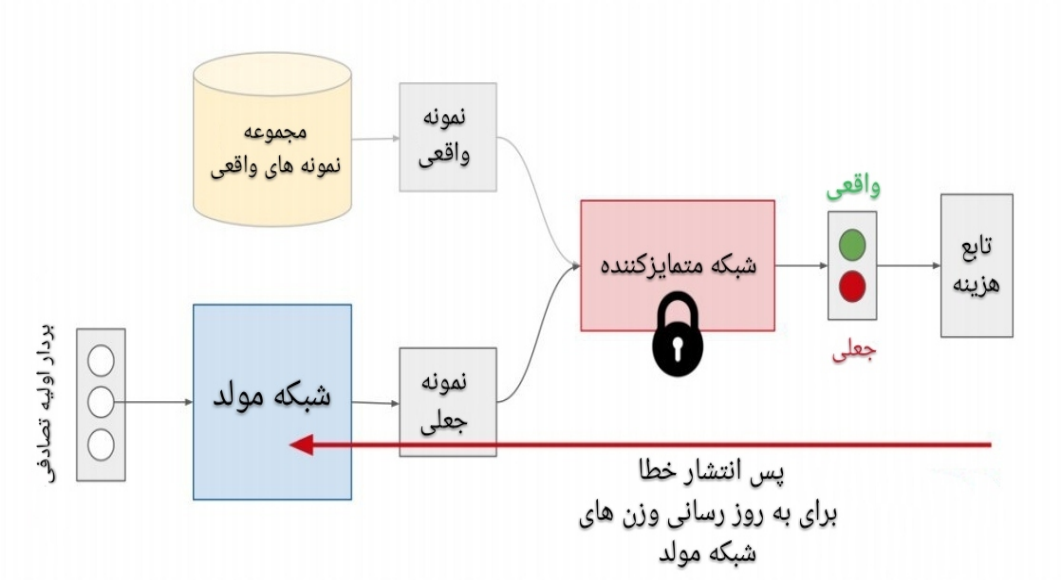
۲: یک دسته‌ی mتایی از نمونه‌های ساخته شده توسط مولد با برچسب صفر به شبکه داده می‌شود.

$$\left[\begin{array}{cccc} x_۱ & x_۲ & x_۳ & \dots & x_m \end{array} \right] \sim P_g(x)$$

وزن‌های شبکه برای کمینه سازی تابع هزینه زیر توسط الگوریتم پس انتشار خطا (۱-۱) به‌هنگام می‌شوند.

$$\mathbf{J}(\mathbf{D}) = -\frac{۱}{۲} E_{z \sim p_z} \left[\log \left(۱ - \mathbf{D}(\mathbf{G}(z)) \right) \right]$$

۲-۴-۲ آموزش شبکه مولد



شکل ۲-۹: آموزش شبکه مولد در GAN

شبکه مولد برای تولید نمونه‌های جعلی از طریق بازخوردی که از شبکه متمایزکننده دریافت می‌کند، آموزش میبیند. این شبکه در تلاش است که نمونه‌های جعلی مشابه با نمونه‌های واقعی تولید کند، به گونه‌ای که شبکه متمایزکننده قادر به تشخیص نمونه‌های جعلی از واقعی نباشد. بر این اساس زمانی که شبکه متمایزکننده در طبقه‌بندی تصاویر جعلی و واقعی فریب بخورد، کار شبکه مولد به درستی انجام شده است [۲۲]. به همین دلیل برای آموزش، هر دو شبکه مورد نیاز است، مانند زمانی که شبکه متمایزکننده را با داده‌های تولید شده توسط مولد، آموزش دادیم (۲-۴-۱). همانند شکل ۲-۹ شبکه متمایزکننده قفل شده و آموزش آن متوقف می‌شود. در این صورت وزن‌های شبکه مولد با قاعده پس انتشار خطا (بخش ۱-۲) به روز رسانی می‌شوند [۱]. برای این منظور باید تابع هزینه $\mathbf{J(G)}$ این شبکه را کمینه کرد. طبق رابطه (۲-۷)، کمینه کردن تابع هزینه $\mathbf{J(G)}$ شبکه مولد با بیشینه کردن تابع هزینه $\mathbf{J(D)}$ شبکه متمایزکننده برابر است و می‌توان تابع هزینه شبکه متمایزکننده را افزایش داد. پس تابع هزینه مورد استفاده در شبکه مولد، همان تابع هزینه شبکه متمایزکننده در رابطه (۲-۹) است که در این بخش باید بیشینه گردد. برای آموزش این بخش فقط از داده‌های تولید شده توسط مولد، استفاده می‌شود، از این رو در رابطه تابع هزینه، قسمت مربوط به داده‌های اصلی، صفر است و رابطه تابع هزینه شبکه مولد به صورت زیر بازنویسی می‌شود [۱]:

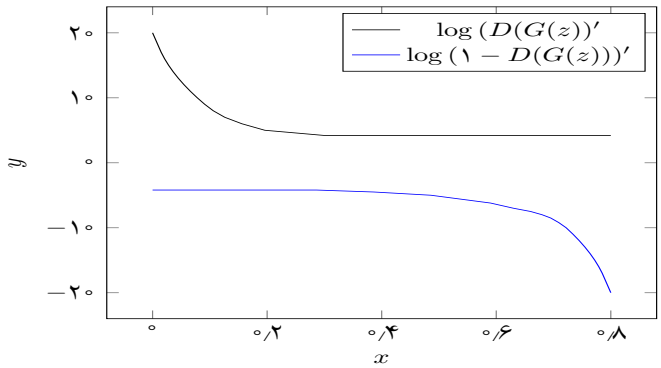
$$\mathbf{J(G)} = E_{z\sim p_z}\left[\log\left(1-\mathbf{D}(\mathbf{G}(z))\right)\right] \qquad (۲-۱۰)$$

[۱] https://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

برای به دست آوردن وزن‌های مناسب شبکه عصبی طبق بخش ۱-۱-۳ به مشتق تابع هزینه نیاز است. مشتق تابع هزینه در شبکه مولد به صورت $-\frac{۱}{۱-\mathbf{D}(\mathbf{G}(z))}$ محاسبه می‌شود که در شکل ۲-۱۰ نمایش داده شده است. با توجه به شکل با شروع آموزش، شبکه متمایزکننده به خوبی داده‌های واقعی را از داده‌های تولیدی متمایز می‌کند یعنی $\mathbf{D}(\mathbf{G}(z)) = ۰$، و شیب گرادیان نزدیک به صفر است بدین معنا که تابع هزینه کمینه نمی‌شود. به عنوان مثال در شکل ۲-۱۱ شبکه مولد رقابتی برای سه تعداد تکرار متفاوت آموزش داده شده است. در هر سه تعداد تکرار، خطای شبکه متمایزکننده به صفر نزدیک می‌شود؛ این بیانگر این است که شبکه متمایزکننده به درستی نمونه‌های واقعی و نمونه‌های تولید شده را طبقه‌بندی کرده است. از طرفی با آموزش بیشتر و بهتر شدن شبکه متمایزکننده، شیب شبکه مولد ازبین می‌رود و این مشکل باعث یادگیری تعداد کمی از وزن‌های θ_g می‌شود. برای حل این مشکل، تابع هزینه شبکه مولد بالا به صورت زیر بازنویسی می‌شود [۱، ۳۱]:

$$\mathbf{J}(\mathbf{G}) = -E_{z\sim p_z}\left[\log\left(\mathbf{D}(\mathbf{G}(z))\right)\right] \qquad (۲-۱۱)$$

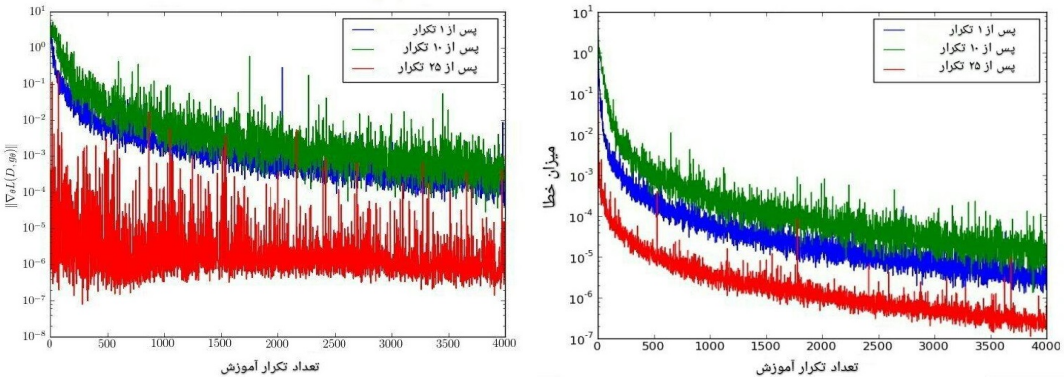
گرادیان این رابطه در شکل ۲-۱۰ با رنگ سیاه مشخص شده است. مقدار تابع هزینه در این رابطه زمانی که $\mathbf{D}(\mathbf{G}(z)) = ۱$ است نیز کاهش می‌یابد و شیب آن زیاد است. این موضوع یادگیری را در همه حالات نشان می‌دهد. با انتخاب این تابع هزینه، بازی بین شبکه مولد و شبکه متمایزکننده، به صورت مجموع-صفر نیست اما این مسئله، مشکلی در چارچوب شبکه مولد رقابتی ایجاد نمی‌کند [۱].



شکل ۲-۱۰: رسم گرادیان توابع هزینه مورد استفاده در آموزش شبکه مولد (روابط (۲-۱۰)و(۲-۱۱))

۲-۴-۲-۱ الگوریتم شبکه عصبی مولد

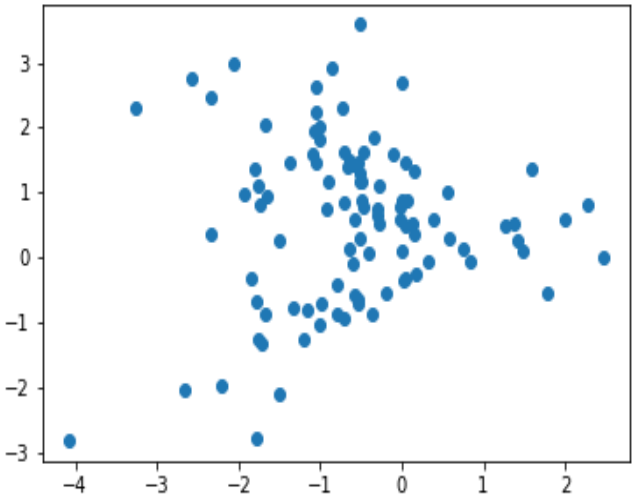
چارچوب کلی آموزش شبکه مولد در الگوریتم ۲-۲ آمده است. بردار نویز تصادفی به شبکه مولد داده می‌شود. این شبکه ابعاد بردار اولیه را به ابعاد نمونه‌ی مورد نظر تبدیل کرده و نمونه‌های جعلی تولید می‌کند. برنامه پایتون



(آ) (ب)

شکل ۲-۱۱: شبکه مولد رقابتی برای تعداد ۱، ۱۰ و ۲۵ بار به صورت مجزا آموزش دیده است. خطای شبکه متمایزکننده در هر سه تعداد تکرار، به صفر نزدیک می‌شود؛ بنابراین این شبکه به درستی طبقه‌بندی می‌کند(آ). با بهتر شدن شبکه متمایزکننده، شیب شبکه مولد ازبین می‌رود و منجر به آموزش کمتر این شبکه می‌شود(ب).

آ-۵ موجود در پیوست، آموزش شبکه مولد را نشان می‌دهد. خروجی این شبکه در شکل ۲-۱۲ آورده شده است. شبکه مولد به تنهایی آموزش دیده است به همین دلیل خروجی مناسبی ندارد. با آموزش شبکه مولد درون یک شبکه مولد رقابتی، نمونه‌هایی تولید می‌شود که شبکه متمایزکننده قادر به تشخیص جعلی بودن آن‌ها نشود. در بخش ۲-۴-۳-۱ توضیحات بیشتری بیان شده و خروجی شبکه مولد رقابتی آورده شده است [۳۰].



شکل ۲-۱۲: نقاط تولید شده در فضای $\mathbb{R}^2$ توسط شبکه مولد که به تنهایی آموزش دیده است.

الگوریتم ۲-۲ الگوریتم آموزش شبکه مولد

ورودی: بردار نویز تصادفی $z \sim N(۰, ۱)$ یا $z \sim U(-۱, ۱)$
خروجی: نمونه‌های جدید $x \sim P_g(x)$

۱: بردار نویز تصادفی زیر به شبکه اعمال می‌شود.

$$\left[\begin{array}{cccc} z_۱ & z_۲ & z_۳ & \dots & z_m \end{array} \right] \sim P_z(z)$$

۲: وزن‌های θ_g شبکه برای کمینه سازی تابع هزینه زیر توسط الگوریتم پس انتشار خطا (۱-۱) به‌هنگام می‌شوند.

$$\mathbf{J}(\mathbf{G}) = -E_{z \sim p_z} \left[\log \left(\mathbf{D}(\mathbf{G}(z)) \right) \right]$$

۲-۴-۳ آموزش شبکه مولد و شبکه متمایزکننده در قالب شبکه GAN

شبکه مولد و شبکه متمایزکننده به صورت جداگانه آموزش می‌بینند. شبکه مولد رقابتی با اجرای متناوب این دو شبکه، شبکه بزرگ‌تری را تولید و بازی کمینه-بیشینه (بخش ۲-۱-۲) بین این دو برقرار می‌کند. بدین صورت که در ابتدا نمونه‌هایی توسط شبکه مولد تولید می‌شود و این نمونه‌ها با برچسب صفر به معنی جعلی بودن به شبکه متمایزکننده داده می‌شود. از طرفی از نمونه‌های واقعی به صورت تصادفی چند نمونه انتخاب شده و با برچسب ۱ به معنی واقعی بودن به شبکه متمایزکننده داده می‌شود. شبکه متمایزکننده روی این نمونه‌ها با برچسب ۰ و ۱ آموزش می‌بیند. زمانی که شبکه متمایزکننده طبقه‌بندی نمونه‌ها را می‌آموزد، آموزش شبکه مولد توسط شبکه GAN متوقف شده است که در شکل ۲-۸ با کشیدن قفل روی این شبکه نشان داده شده است. پس از آن، آموزش شبکه متمایزکننده توسط شبکه GAN متوقف شده (همانند شکل ۲-۹ قفل روی شبکه متمایزکننده قرار می‌گیرد) و شبکه مولد آموزش می‌بیند. برای آموزش این شبکه، به شبکه متمایزکننده نیاز است. نمونه‌های تولید شده توسط شبکه مولد به شبکه متمایزکننده داده می‌شود و به دلیل آنکه شبکه متمایزکننده قادر به طبقه‌بندی نمونه‌های جعلی و واقعی است، این نمونه را به عنوان نمونه جعلی شناسایی کرده و برچسب صفر می‌دهد. میزان خطای شبکه مولد توسط تابع هزینه‌ی این شبکه محاسبه شده و توسط قائده‌ی پس انتشار خطا به شبکه مولد داده می‌شود و این شبکه نمونه‌های جدیدی می‌سازد که شبکه متمایزکننده را فریب دهد. سپس مجدداً آموزش شبکه مولد متوقف شده و شبکه متمایزکننده آموزش می‌بینند. این روند تا رسیدن هر دو شبکه به تعادل نش ادامه می‌یابد. اگر هر کدام از این شبکه‌ها آموزش کمتری ببینند، نسبت به شبکه‌ی دیگر ضعیف‌تر شده و فریب می‌خورند. در این صورت نمونه‌های مدنظر تولید نشده و کیفیت لازم را نخواهند داشت [۲۰]. در آموزش کلی شبکه، تابع هزینه [۱] $\mathbf{V}$ به

[۱] Cost

صورت کمینه-بیشینه (۲-۱-۲) تعریف می‌شود:

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \mathbf{V}(\mathbf{G}, \mathbf{D}) \qquad (۱۲-۲)$$

با توجه به آموزش‌های شبکه مولد و شبکه متمایزکننده، تابع هزینه شبکه مولد رقابتی به صورت زیر خواهد بود:

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \mathbf{V}(\mathbf{G}, \mathbf{D}) = \frac{۱}{۲} E_{x \sim p_{data}} [\log \mathbf{D}(x)] + \frac{۱}{۲} E_{z \sim p_z} [\log (۱ - \mathbf{D}(\mathbf{G}(z)))] \qquad (۱۳-۲)$$

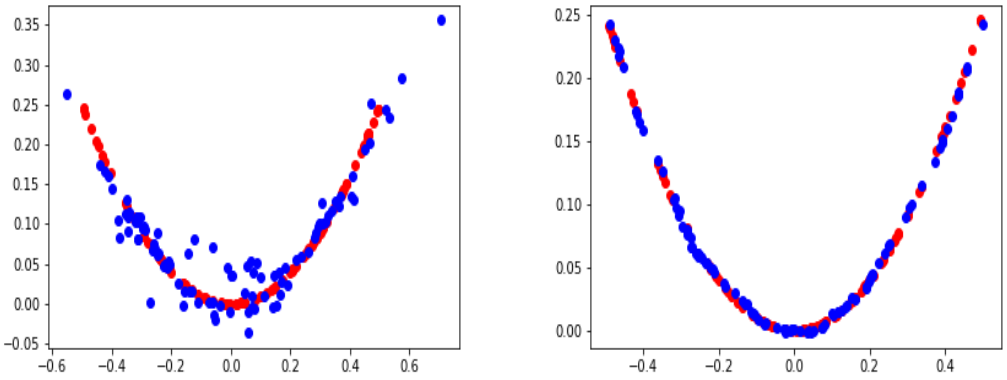
در این تابع شبکه متمایزکننده سعی می‌کند تابع هدف را بیشینه کند و بالعکس، شبکه مولد سعی در کمینه کردن این تابع دارد [۱]. تابع هزینه $\mathbf{V}(\mathbf{G}, \mathbf{D})$ منفی تابع هزینه شبکه متمایزکننده (۲-۹) است. در بخش ۲-۴-۱ بیان شد که شبکه متمایزکننده درصدد کمینه کردن تابع $\mathbf{J}(\mathbf{D})$ است پس تابع $\mathbf{V}(\mathbf{G}, \mathbf{D})$ را بیشینه می‌کند. از طرف دیگر در بخش ۲-۴-۲ بیان شد که شبکه مولد برای کمینه کردن تابع هزینه خود می‌تواند تابع هزینه شبکه متمایزکننده را بیشینه کند. از این رو شبکه مولد تابع هزینه $\mathbf{V}(\mathbf{G}, \mathbf{D})$ را کمینه می‌کند.

۲-۴-۳-۱ الگوریتم و برنامه‌های شبکه مولد رقابتی

چارچوب کلی آموزش شبکه مولد رقابتی در الگوریتم ۲-۳ آمده است. در ابتدا تعدادی از نمونه‌های واقعی به صورت تصادفی انتخاب شده و با برچسب ۱ به شبکه متمایزکننده داده می‌شود. سپس شبکه مولد با بردارهای تصادفی اولیه، نمونه‌هایی را تولید می‌کند. این نمونه‌ها با برچسب صفر به شبکه متمایزکننده برای آموزش داده می‌شود. سپس آموزش شبکه متمایزکننده توقف می‌یابد و شبکه مولد نمونه‌های جدیدی تولید می‌کند. نمونه‌های جدید به همراه نمونه‌های واقعی با برچسب ۱ به شبکه متمایزکننده داده می‌شود. این شبکه با آموزش قبلی خود، نمونه‌های ساخته شده را از نمونه‌های واقعی طبقه‌بندی می‌کند. شبکه مولد که قادر به فریب شبکه متمایزکننده نبوده است، توسط قاعده پس‌انتشار خطا (بخش ۱-۲) به‌هنگام می‌شود.

آموزش شبکه متمایزکننده و شبکه مولد به صورت مجزا به ترتیب در برنامه‌های آ-۴ و آ-۵ بیان شده است. برنامه‌ی آ-۶ دو شبکه متمایزکننده و مولد بیان شده را در قالب یک شبکه مولد رقابتی آموزش می‌دهد. این برنامه نقاط جدید روی تابع $x^۲$ به صورتی که $x \in \mathbb{R}$ باشد را تولید می‌کند. در شکل ۲-۱۲ شبکه مولد برای ساخت نقاط روی تابع $x^۲$ به تنهایی آموزش داده شد اما خروجی مناسب نبود. با آموزش همان شبکه در قالب شبکه مولد رقابتی خروجی تولید شده است که شبکه متمایزکننده قادر به تشخیص جعلی بودن آن‌ها نیست. خروجی شبکه مولد رقابتی در شکل ۲-۱۳ نشان داده شده است. در این شکل نقاط با رنگ قرمز، نقاط واقعی را نشان می‌دهند

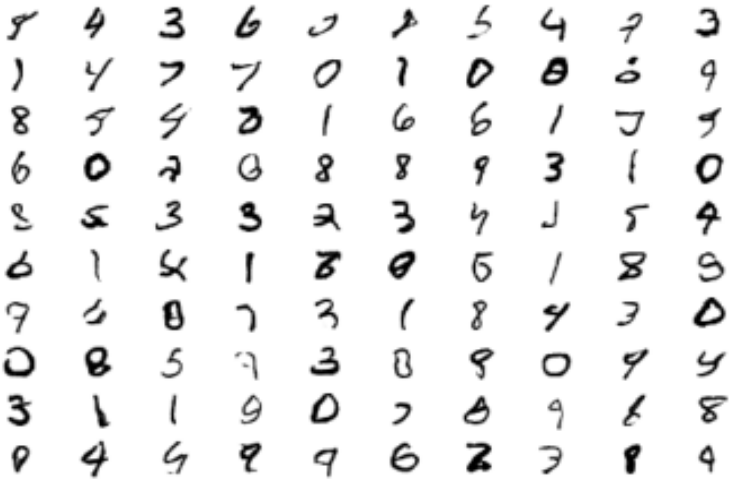
و نقاط آبی توسط مولد تولید شده است. بر طبق شکل نقاط تولید شده پس از ۲۰۰۰ تکرار، کمی خطا دارند بدین معنا که نقاطی تولید شده‌اند که روی تابع x^2 قرار ندارند اما نقاط تولیدی پس از ۱۰۰۰۰ تکرار، کاملا روی تابع قرار گرفته‌اند [۳۰].



(آ) نقاط تولید شده پس از ۱۰۰۰۰ تکرار (ب) نقاط تولید شده پس از ۲۰۰۰ تکرار

شکل ۲-۱۳: تولید نقاط جدید روی تابع x^2 توسط شبکه GAN. نقاط قرمز، نمونه‌های واقعی و نقاط آبی، نقاط تولید شده را نشان می‌دهند.

در برنامه‌ی آ-۷ از شبکه مولد رقابتی برای تولید تصاویر ارقام استفاده شده است. شبکه روی مجموعه داده‌ی ارقام دست‌نویس انگلیسی MNIST[1] آموزش دیده است. این مجموعه ارقام شامل ۷۰۰۰۰ تصویر ۲۸ × ۲۸ از رقم‌های ۰ تا ۹ است؛ تعداد ۶۰۰۰۰ برای آموزش و تعداد ۱۰۰۰۰ برای تست و آزمایش وجود دارد. نتیجه این شبکه مولد رقابتی پس از ۵۰۰ تکرار در شکل ۲-۱۴ آمده است [۲۸].



شکل ۲-۱۴: تصاویر تولید شده توسط GAN با آموزش روی مجموعه داده MNIST

[1] Modified National Institute of Standards and Technology database

الگوریتم ۲-۳ الگوریتم آموزش شبکه مولد رقابتی [۱]

ورودی: نمونه‌های آموزشی و برچسب‌های آن‌ها
خروجی: نمونه‌های جدید مشابه با نمونه‌های آمورشی

۱: مراحل زیر را برای k مرتبه تکرار کنید.
- m نمونه‌ی تصادفی از فضای نهفته انتخاب می‌شود.

$$\left[\begin{array}{cccccc} z_۱ & z_۲ & z_۳ & \dots & z_m \end{array} \right] \sim P_z(z)$$

- m نمونه‌ی تصادفی از نمونه‌های واقعی انتخاب می‌شود.

$$\left[\begin{array}{cccccc} x_۱ & x_۲ & x_۳ & \dots & x_m \end{array} \right] \sim P_{data}(x)$$

- مشتق تابع هزینه شبکه متمایزکننده به صورت زیر محاسبه می‌شود و وزن‌های این شبکه توسط الگوریتم پس انتشار خطا (۱-۱) به‌هنگام می‌شوند.

$$\nabla_{\theta_d} \frac{۱}{m} \sum_{i=۱}^{m} [\log \mathbf{D}(x_i) + \log (۱ - \mathbf{D}(\mathbf{G}(z_i)))]$$

۲: m نمونه‌ی تصادفی از فضای نهفته(اولیه) انتخاب می‌شود.

$$\left[\begin{array}{cccccc} z_۱ & z_۲ & z_۳ & \dots & z_m \end{array} \right] \sim P_z(z)$$

۳: مشتق تابع هزینه شبکه متمایزکننده به صورت زیر محاسبه می‌شودد و وزن‌های این شبکه توسط الگوریتم پس انتشار خطا (۱-۱) به‌هنگام می‌شوند.

$$\nabla_{\theta_g} \frac{۱}{m} \sum_{i=۱}^{m} \log (۱ - \mathbf{D}(\mathbf{G}(z_i)))$$

در بازی مجموع-صفر بین دو بازیکن، بُرد یک بازیکن برابر با باخت بازیکن دیگر است. از این رو هر دو یک جواب بهینه دارند. بیشینه کردن جواب توسط بازیکن اول، برابر با کمینه کردن جواب بازیکن دوم است. به این قضیه کمینه-بیشینه گویند.

۲-۵ انواع شبکه مولد رقابتی

قابلیت انعطاف‌پذیری در شبکه‌های مولد رقابتی باعث شده است که در طی سال‌های گذشته روی این موضوع تحقیقات بسیار شود و نسخه‌های توسعه یافته از این شبکه را ارائه دهند. در جدول ۲-۱ برخی از شبکه‌های مولد رقابتی ساخته شده از سال ۲۰۱۴ تا ۲۰۱۹ بیان شده است [۳۲]. در فصل بعدی به شبکه مولد رقابتی وَسِراِستین پرداخته خواهد شد.

جدول ۲-۱: انواع شبکه مولد رقابتی در طول سال‌های اخیر

سال تولید	نام شبکه	ارائه دهنده
۲۰۱۴	شبکه مولد رقابتی [۲۶] (Generative Adversaial Network) (GAN)	یان گودفلو (Ian Goodfellow al.)
	شبکه مولد رقابتی شرطی [۳۳] (Conditional Generative Adversaial Network) (Conditional GAN)	مهدی میرزا (Mehdi Mirza al.)
۲۰۱۵	هرم لاپلاس از شبکه مولد رقابتی [۳۴] (Laplacian Pyramid of Generative Adversarial Networks) (LAPGAN)	دنتون (Emily Denton al.)
	شبکه مولد رقابتی پیچشی عمیق [۳۵] (Deep Convolutional Generative Adversarial Networks) (DCGAN)	(Alec Radford al.)
۲۰۱۷	شبکه مولد رقابتی Pix2Pix [۳۶] (Pix2Pix Generative Adversarial Network) (Pix2pixGAN)	(Phillip Isola al.)
	طبقه‌بندی کمکی شبکه مولد رقابتی [۳۷] (Auxiliary Classifier Generative Adversarial Network) (AC-GAN)	(Augustus Odena al.)
	شبکه مولد رقابتی وَسِراِستین [۲] (Wasserstein Generative Adversarial Network) (WGAN)	مارتین آرجوزکی (Martin Arjovsky al.)
	شبکه مولد رقابتی چرخه‌ای [۳۸] (Cycle Generative Adversarial Network) (CycleGAN)	(Jun-Yan Zhu al.)
۲۰۱۸	شبکه مولد رقابتی ستاره [۳۹] (Star Generative Adversarial Network) (StarGAN)	(Yunjey Choi al.)
	شبکه مولد رقابتی بزرگ [۴۰] (Big Generative Adversarial Network) (BigGAN)	(Andrew Brock al.)
۲۰۱۹	شبکه مولد رقابتی سبک [۴۱] (Style Generative Adversarial Network) (StyleGAN)	(Tero Karras al.)

فصل ۳

شبکه مولد رقابتی وَسِراِستین

در فصل قبل با شبکه مولد رقابتی و نحوه‌ی آموزش آن آشنا شدیم. آموزش این شبکه شامل آموزش پی در پی شبکه مولد و شبکه متمایزکننده در یک بازی مجموع-صفر و همچنین آموزش مداوم تا رسیدن به نقطه تعادل بین این دو شبکه است. نوعی از شبکه‌های مولد رقابتی با نام شبکه مولد رقابتی وَسِراِستین [۲] در سال ۲۰۱۷ توسط مارتین آرجُوزکی [۳] و همکارانش ارائه شد. شبکه مولد رقابتی وَسِراِستین یا WGAN توسعه‌ای از شبکه‌های مولد رقابتی است و تابع هزینه جدیدی ارائه می‌دهد که با کیفیت تصاویر تولید شده ارتباط دارد. این شبکه معیارهای روشن‌تری برای متوقف شدن آموزش ارائه می‌دهد. آموزش شبکه‌های WGAN نیاز به حفظ تعادل نش در آموزش شبکه مولد و شبکه متمایزکننده ندارد. در این شبکه به طراحی و معماری دقیق شبکه‌ها در کنار یکدیگر احتیاج نیست. بدین صورت که نیاز به آموزش پی در پی و یکسان شبکه مولد و شبکه متمایزکننده نیست و شبکه‌ها می‌توانند تعداد دفعات بیشتری نسبت به یکدیگر آموزش ببینند. به همین دلیل شبکه WGAN به تغییراتی در ساختار و نحوه‌ی آموزش شبکه‌های مولد و متمایزکننده نیاز دارد. این شبکه فاصله‌ی وَسِراِستین $\mathbf{W}$ (بخش ۳-۱-۱) میان توزیع داده‌های واقعی و توزیع داده‌های تولید شده را به دست می‌آورد. یکی از مزایای عملی این شبکه، پیوستگی این فاصله در آموزش شبکه منتقد به صورت بهینه است؛ بدین معنا که شبکه متمایزکننده که در این جا با کمی تغییرات به عنوان شبکه منتقد شناخته می‌شود، قادر است تا هنگام بهینه شدن، آموزش ببیند [۲، ۲۲].

در این فصل ابتدا به بررسی چند فاصله، همگرایی و مطالب مورد نیاز در زمینه شبکه مولد رقابتی وَسِراِستین می‌پردازیم. در ادامه قضایای مورد نیاز بیان شده و سپس روش آموزش شبکه مولد رقابتی وَسِراِستین را بررسی خواهیم کرد.

---

[۲] Wasserstein Generative Adversial Network (WGAN) [۳] Martin Arjovsky

۳-۱ پیش نیازها

برای بیان و اثبات مطالب این فصل نیاز به قضایا و تعاریفی است که در این بخش به آن‌ها می‌پردازیم.

۳-۱-۱ انواع فاصله

تابع هدف فاصله‌ی بین توزیع نمونه‌های تولید شده توسط مدل و توزیع نمونه‌های واقعی را به دست می‌آورد. برای شبکه‌های مولد رقابتی توابع هدف متفاوتی می‌توان تعریف نمود. در این توابع هدف می‌توان از معیارهای سنجش دو توزیع آماری همچون جیسون شانون [۲۶]، واگرایی‌های بیان شده در مرجع [۴۲] و برخی ترکیبات مرجع [۴۳] استفاده نمود. توابع هدف به دنبال کمینه سازی خطاهای مدل هستند که برای این منظور پارامترهای مدل به صورت بهینه به‌هنگام می‌شوند. برای بهینه سازی پارامتر θ مطلوب است که توزیع مدل P_θ توسط روشی انجام شود که نگاشت $\theta \rightarrow P_\theta$ پیوسته باشد؛ پیوستگی بدین معنا که اگر دنباله‌ای از پارامترهای θ_t به θ همگرا شدند ($\theta_t \rightarrow \theta$)، توزیع P_{θ_t} به P_θ همگرا شود ($P_{\theta_t} \rightarrow P_\theta$). همگرایی توزیع P_{θ_t} به نحوه‌ی محاسبه فاصله میان توزیع‌ها بستگی دارد؛ هرچه فاصله‌ی میان توزیع نمونه‌ی ساخته شده توسط مدل و توزیع نمونه‌ی واقعی کمتر باشد، امکان تعریف یک نگاشت پیوسته از فضای θ به فضای P_θ، به دلیل سادگی همگرایی توزیع، ساده‌تر خواهد بود. اهمیت نگاشت $\theta \rightarrow P_\theta$ به دلیل این است که اگر ρ مفهوم فاصله بین دو توزیع باشد، تابع هزینه $\theta \rightarrow \rho(P_\theta, P_{data})$ باید پیوسته باشد و این معادل است با این که در هنگام استفاده از فاصله‌ی بین توزیع‌ها (ρ)، نگاشت $\theta \rightarrow P_\theta$ پیوسته باشد [۲].

در اکثر این توابع هدف، میزان تفاوت یا شباهت میان نمونه واقعی و نمونه تولید شده مدنظر است. روش‌های مختلفی به منظور سنجش میزان شباهت میان توزیع نمونه‌های واقعی و توزیع مدل وجود دارد. در این بخش چند نمونه از این روش‌ها برای به دست آوردن میزان فاصله و همگرایی $\rho(P_{data}, P_\theta)$ بیان شده است. تفاوت اساسی میان این فاصله‌ها، تأثیر آن‌ها بر همگرایی دنباله‌ای از توزیع‌های احتمالاتی است. قضیه ۳-۱-۸ به همگرایی اشاره می‌کند. در این قضیه بستگی دارد که فاصله از چه روشی محاسبه شود.

با در نظر گرفتن $\mathcal{X}$ به عنوان مجموعه متریک پیوسته [۱] (مانند مجموعه فضای تصاویر $[0,1]^d$) و Σ به عنوان مجموعه تمام زیرمجموعه‌های بورل [۲] از مجموعه $\mathcal{X}$، می‌توان پیش‌آمدهای احتمالی روی $\mathcal{X}$ را با $Prop(\mathcal{X})$ نشان داد. با در نظر گرفتن این موارد، فاصله و همگرایی بین دو توزیع $P_{data}, P_g \in Prop(\mathcal{X})$ به صورت‌های زیر محاسبه می‌شود [۲]:

---

[۱] Compact Metric Set [۲] Borel subsets

- **فاصله تغییرات کل** [۱] **(TV)** .

$$\delta(P_{data}, P_g) = \sup_{A \in \sum} \mid P_{data}(A) - P_g(A) \mid \qquad (۱-۳)$$

- **واگرایی** [۲] **KL** . با فرض این که توزیع‌های P_{data} و P_g پیوسته هستند، این روش به صورت کمینه سازی رابطه زیر خواهد بود:

$$KL(P_{data} \| P_g) = \int \log \left(\frac{P_{data}(x)}{P_g(x)} \right) P_{data}(x) d\mu(x) \qquad (۲-۳)$$

در این رابطه چگالی با تأثیر یک معیار ثابت μ روی $\mathcal{X}$ به دست می‌آید. واگرایی KL در کمینه کردن $P_g = P_{data}$ منحصر به فرد است و برای بهینه سازی آن به اطلاعاتی در مورد $P_{data}(x)$ نیاز نیست. با این حال این واگرایی بین P_{data} و P_g متقارن نیست [۳۱].

- **واگرایی جیسون-شانون** [۳] **(JS)** .

$$JS(P_{data} \| P_g) = KL(P_{data} \| P_m) + KL(P_g \| P_m) \qquad (۳-۳)$$

در این رابطه مقدار P_m از رابطه $P_m = \frac{P_{data} + P_g}{۲}$ به دست می‌آید. واگرایی جیسون-شانون متقارن است و با انتخاب $\mu = P_m$ مرسوم است.

- **فاصله وَسِراستین W یا EMD** [۴] .

$$\begin{aligned} \mathbf{W}(P_{data}, P_g) &= \inf_{\gamma \in \Pi(P_{data}, P_g)} \int_{\mathcal{X} \times \mathcal{X}} \|x - y\| d\gamma(x, y) \\ &= \inf_{\gamma \in \Pi(P_{data}, P_g)} E_{(x,y) \sim \gamma} \left[\|x - y\| \right] \end{aligned} \qquad (۴-۳)$$

در این رابطه $\Pi(P_{data}, P_g)$ مجموعه‌ای از توزیع‌های مشترک $\gamma(x, y)$ را نشان می‌دهد. $\gamma(x, y)$ نشان دهنده این است که برای تبدیل توزیع P_{data} به P_g، چه مقدار جرم از x به y منتقل شود. فاصله وَسِراستین $\mathbf{W}$ هزینه‌ی انتقال بهینه‌ی نگاشت را محاسبه می‌کند.

---

[۱] The Total Variation distance [۲] The Kullback-Leibler divergence [۳] The Jensen-Shannon divergence [۴] Wasserstein distance (The Earth-Mover distance or Transportation Metric)

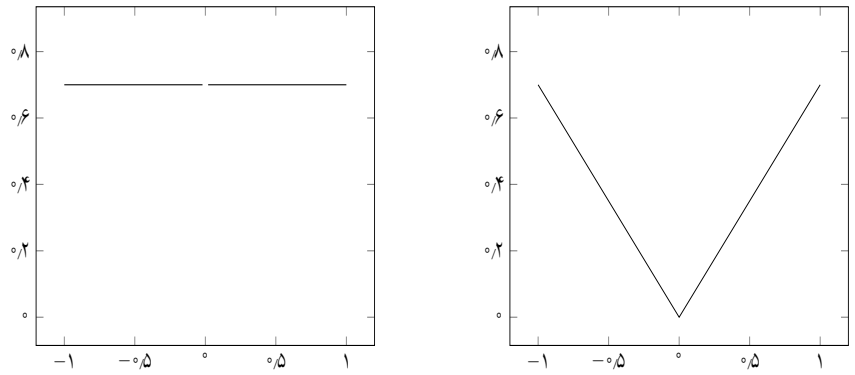
شبکه WGAN فاصله‌ی توزیع نمونه‌ی واقعی مشاهده شده در مجموعه داده آموزشی و توزیع نمونه‌های مشاهده شده از مجموعه نمونه‌های تولید شده را محاسبه می‌کند.

مثال ساده‌ی زیر نشان می‌دهد که دنباله‌های ساده توزیع احتمال با فاصله‌ی وَسِراِستین $\mathbf{W}$ همگرا می‌شوند و با بقیه فواصل و واگرایی‌های تعریف شده همگرا نمی‌شود.

مثال ۳-۱-۱ (یادگیری خطوط موازی). فرض کنید $z \sim U[۰, ۱]$ توزیع یکنواخت در فاصله واحد باشد. $P_۰$ توزیع یکنواخت از $(۰, z) \in R^۲$ روی یک خط مستقیم عمودی که از مبدأ مختصات عبور می‌کند را در نظر بگیرید. اکنون فرض کنید با داشتن یک پارامتر واقعی θ رابطه $g_\theta(z) = (\theta, z)$ برقرار باشد. در این حالت موارد زیر برقرار است:

$$
\begin{aligned}
&- \mathbf{W}(P_۰, P_\theta) = |\theta| \\
&- JS(P_۰, P_\theta) = \begin{cases} \log ۲ & \text{if } \theta \neq ۰ \\ ۰ & \text{if } \theta = ۰ \end{cases} \\
&- KL(P_۰, P_\theta) = KL(P_\theta, P_۰) = \begin{cases} +\infty & \text{if } \theta \neq ۰ \\ ۰ & \text{if } \theta = ۰ \end{cases} \\
&- \delta(P_۰, P_\theta) = \begin{cases} ۱ & \text{if } \theta \neq ۰ \\ ۰ & \text{if } \theta = ۰ \end{cases}
\end{aligned}
$$

در فاصله‌ی وَسِراِستین $\mathbf{W}$ زمانی که $\theta_t \to ۰$ دنباله‌ی $(P_{\theta_t})_{t \in N}$ به $P_۰$ همگرا می‌شود. در نتیجه روش تابع بهینه سازی سریع‌ترین شیب روی فاصله‌ی وَسِراِستین $\mathbf{W}$ یک توزیع احتمالاتی را روی منیفولد با ابعاد کم، یاد می‌گیرد. این آموزش با فاصله‌ی TV، واگرایی KL و جیسون شانون JS امکان‌پذیر نیست زیرا پیوسته نیستند و همگرا نمی‌شوند. در شکل ۳-۱ همگرا شدن فاصله‌ی وَسِراِستین $\mathbf{W}$ و واگرا شدن JS نشان داده شده است. در قضیه ۳-۱-۶ استحکام نسبی توپولوژی‌های ناشی از فواصل و واگرایی‌ها بیان شده است. بر طبق آن واگرایی KL، جیسون-شانون JS و فاصله‌ی تغییرات کل TV از استحکام بالاتری نسبت به فاصله‌ی وَسِراِستین $\mathbf{W}$ یا همان فاصله‌ی EMD برخوردار است. فاصله‌ی وَسِراِستین از دیگر فاصله‌های بیان شده ضعیف‌تر است، با این حال $\mathbf{W}(P_{data}, P_\theta)$ یک تابع هزینه پیوسته روی θ است. این ادعا در قضیه ۳-۱-۴ بیان و اثبات شده است. از این رو می‌توان از فاصله وَسِراِستین در شبکه‌های عصبی استفاده نمود. در نتیجه‌گیری ۳-۱-۵ بیان شده است که می‌توان با به حداقل رساندن فاصله وَسِراِستین، شبکه عصبی را آموزش داد [۲].



(ب) واگرایی جیسون-شانون JS

(آ) همگرایی فاصله‌ی وَسِراستین **W**

شکل ۳-۱: نمودار فاصله $\rho(P_\theta, P_0)$ بر حسب θ. فاصله در نمودار سمت چپ **(JS)** پیوسته نیست و شیب قابل ملاحظه ندارد. فاصله در نمودار سمت راست **(W)** پیوسته است و شیب مناسبی دارد.

۳-۱-۲ تعاریف و قضایا

تعریف ۳-۱-۲ (لیپشیتز). فرض کنید (X, d_X) و (Y, d_Y) دو فضای متریک باشد به گونه‌ای که d_X متریک موجود در مجموعه‌ی X و d_Y متریک موجود در مجموعه‌ی Y باشد. در این صورت اگر یک ثابت واقعی $K \geq 0$ وجود داشته باشد که برای تمام $x_1, x_2 \in X$ رابطه زیر برقرار باشد.

$$d_Y\left(f(x_1), f(x_2)\right) \leq K d_X(x_1, x_2)$$

تابع $f : X \to Y$ لیپشیتز با ثابت K [1] نامیده می‌شود. اگر عدد ۱ برای مقدار K در نظر گرفته شود $(K = 1)$، تابع f را لیپشیتز با ثابت ۱ [2] می‌نامند و رابطه بالا به صورت زیر بازنویسی می‌شود:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|$$

تعریف ۳-۱-۳ (لیپشیتز محلی). فرض کنید X یک فضای متریک پیوسته‌ی محلی باشد و f تابع لیپشیتز پیوسته باشد؛ بدین منظور شرایط موجود در قضیه ۳-۱-۲ برقرار است. اگر برای هر $x \in X$ یک همسایگی U از x وجود داشته باشد به طوری که تابع f محدود به این همسایگی باشد. در این صورت تابع f لیپشیتز محلی پیوسته است اگر و تنها اگر این تابع در هر زیرمجموعه پیوسته از X به صورت پیوسته باشد.

قضیه ۳-۱-۴. فرض کنید P_{data} توزیع ثابت روی $\mathcal{X}$ و z یک متغیر تصادفی (به طور مثال گوسی) از فضای

[1] K-Lipschitz [2] 1-Lipschitz

$\mathcal{Z}$ باشد. در نظر بگیرید $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ را به عنوان تابعی که برای z و θ به ترتیب مختصات اول و دوم، $g(z)$ نشان می‌دهد. در این صورت با در نظر گرفتن P_θ به عنوان توزیعی از $g_\theta(z)$ موارد زیر را داریم [۲۸]:

- اگر g تابع پیوسته روی θ باشد سپس $\mathbf{W}(P_{data}, P_\theta)$ وجود دارد.
- اگر g تابع لیپشیتز محلی بین فضاهای برداری با بُعد محدود باشد و ارزیابی $g_\theta(z)$ روی مختصات (z, θ) تعریف شود و برای توزیع احتمالاتی خاص p روی $\mathcal{Z}$، ثابت لیپشیتز محلی $L(\theta, z)$ به صورت زیر باشد

$$E_{z\sim p}\left[L(\theta, z)\right] < +\infty$$

سپس $\mathbf{W}(P_{data}, P_\theta)$ در همه‌جا تابعی پیوسته با مقادیر تقریبا متفاوت است.

- دو مورد قبل برای واگرایی جیسون-شانون $JS(P_{data}, P_\theta)$ و تمام واگرایی‌های KL برقرار نیست.

برهان. فرض کنید θ و θ' دو بردار پارامتر در R^d باشند. در ابتدا مقدار $\mathbf{W}(P_\theta, P_{\theta'})$ را به دست می‌آوریم. بدین منظور از عنصر $\gamma \in \Pi(P_\theta, P_{\theta'})$ برای اتصال $g_\theta(z)$ و $g_{\theta'}(z)$ استفاده می‌شود. بر طبق تعریف فاصله وَسِراِستین (۳-۴) رابطه زیر برقرار است

$$\begin{aligned}\mathbf{W}(P_\theta, P_{\theta'}) &\leq \int_{\mathcal{X}\times\mathcal{X}} \|x - y\| d\gamma \\ &= E_{(x,y)\sim\gamma}\left[\, \|x - y\| \,\right] \\ &= E_z\left[\, \|g_\theta(z) - g_{\theta'}(z)\| \,\right]\end{aligned}$$

اگر g روی θ پیوسته باشد و $\theta \to \theta'$ سپس رابطه $g_\theta(z) \xrightarrow{\theta \to \theta'} g_{\theta'}(z)$ برقرار است، بنابراین $\|g_\theta - g_{\theta'}\| \to 0$ به عنوان تابعی از z وجود دارد. از آنجایی که $\mathcal{X}$ پیوسته است، فاصله‌ی هر دو عنصر موجود در آن به طور یکسان توسط مقدار ثابت M کراندار می‌شود. بنابراین برای تمام θ و z ها رابطه $\|g_\theta(z) - g_{\theta'}(z)\| \leq M$ برقرار است. طبق رابطه همگرایی محدود رابطه زیر را داریم.

$$\mathbf{W}(P_\theta, P_{\theta'}) \leq E_z\left(\, \|g_\theta(z) - g_{\theta'}(z)\| \,\right) \xrightarrow{\theta \to \theta'} 0$$

سپس رابطه زیر پیوستگی $\mathbf{W}(P_r, P_\theta)$ را اثبات می‌کند.

$$\|\mathbf{W}(P_r, P_\theta) - \mathbf{W}(P_r, P_{\theta'})\| \leq \mathbf{W}(P_\theta, P_{\theta'}) \xrightarrow{\theta \to \theta'} 0$$

اکنون فرض کنید g لیپشیتز محلی باشد. سپس برای یک جفت معین (θ, z)، یک مقدار ثابت $L(\theta, z)$ و یک مجموعه باز U به صورتی که $(\theta, z) \in U$ و در این صورت برای هر $(\theta', z') \in U$ رابطه زیر برقرار است.

$$\|g_\theta(z) - g'_\theta(z')\| \leq L(\theta, z)\,(\|\theta - \theta'\| + |z - z'|)$$

با در نظر گرفتن $z' = z$ و $(\theta', z) \in U$ رابطه زیر به وجود می‌آید.

$$E_z\left(\,\|g_\theta(z) - g_{\theta'}(z)\|\,\right) \leq \|\theta - \theta'\|\, E_z\left[L(\theta, z)\right]$$

بنابراین رابطه $U_\theta = \{\theta' \mid (\theta', z) \in U\}$ را تعریف می‌کنیم. به وضوح مشخص است که اگر U مجموعه باز است پس U_θ نیز مجموعه باز است. علاوه بر این اگر g تابع لیپشیتز محلی بین فضاهای برداری با بُعد محدود باشد و ارزیابی $g_\theta(z)$ روی مختصات (z, θ) تعریف شود و برای توزیع احتمالاتی خاص p روی $\mathcal{Z}$، ثابت لیپشیتز محلی$L(\theta, z)$ به صورت $E_{z\sim p}\left[L(\theta, z)\right] < +\infty$ باشد، سپس می‌توان $L_\theta = E_z\left[L(\theta, z)\right]$ را تعریف و به دست آورد.

$$\|\mathbf{W}(P_r, P_\theta) - \mathbf{W}(P_r, P_{\theta'})\| \leq \mathbf{W}(P_\theta, P_{\theta'}) \leq L_\theta \leq \|\theta - \theta'\|$$

برای همه Uها $\mathbf{W}(P_r, P_\theta)$ لیپشیتز محلی است. بنابراین در همه جا پیوسته است و طبق قضیه Radamacher تقریبا در همه جا متفاوت است.

□

نتیجه ۳-۱-۵. فرض کنید g_θ هر شبکه عصبی پیشرو [۱] با پارامترهای θ باشد و $P(z)$ احتمالی از z است به گونه‌ای که رابطه $E_{z\sim P(z)}\,\|z\| < \infty$ برقرار باشد. اگر g_θ تابع لیپشیتز محلی بین فضاهای برداری با بُعد محدود باشد و ارزیابی $g_\theta(z)$ روی مختصات (z, θ) تعریف شود و برای توزیع احتمالاتی خاص p روی $\mathcal{Z}$،

[۱] Feed-Forward Network

ثابت لیپشیتز محلی $L(\theta, z)$ به صورت زیر باشد:

$$E_{z\sim p}\left[L(\theta, z)\right] < +\infty$$

سپس $\mathbf{W}(P_{data}, P_\theta)$ در همه‌جا تابعی پیوسته با مقادیر تقریبا متفاوت است. اثبات این مورد در مرجع *[۲]* بیان شده است.

قضیه ۳-۱-۶. فرض کنید P یک توزیع در فضای پیوسته $\mathcal{X}$ و $(P_n)_{n\in N}$ دنباله‌ای از توزیع‌ها روی $\mathcal{X}$ باشد. سپس با در نظر گرفتن تمام محدودیت‌ها به صورت $n \to \infty$، موارد زیر برقرار هستند *[۲]*:

۱. اظهارات زیر معادل هستند

- $\delta(P_n, P) \to ۰$ ؛ δ فاصله تغییرات کل بیان شده در بخش ۳-۱-۱ است.
- $JS(P_n, P) \to ۰$ ؛ JS واگرایی جیسون-شانون بیان شده در بخش ۳-۱-۱ است.

۲. اظهارات زیر معادل هستند

- $\mathbf{W}(P_n, P) \to ۰$ ؛ فاصله‌ی وَسِراِستین (بخش ۳-۱-۱)
- $P_n \xrightarrow{D} P$ ؛ نماد $\xrightarrow{D}$ نشان‌دهنده همگرایی در توزیع برای متغیرهای تصادفی است.

۳. موارد زیر دلالت بر عبارت اول دارد

$$KL(P_n\|P) \to ۰ \quad , \quad KL(P\|P_n) \to ۰$$

منظور از KL واگرایی بیان شده در بخش ۳-۱-۱ است.

۴. عبارت اول دلالت بر عبارت دوم دارد. اثبات این قضیه در مرجع *[۲]* بیان شده است.

قضیه ۳-۱-۷. فرض کنید P_{data} توزیع نمونه‌های واقعی باشد و P_θ توزیعی از $g_\theta(z)$ باشد که z متغیر تصادفی با چگالی P و g_θ تابع لیپشیتز محلی (۳-۱-۳) بین فضاهای برداری با بُعد محدود به صورت $g : \mathcal{Z}\times\mathbb{R}^d \to \mathcal{X}$

است. سپس جواب $f : \mathcal{X} \to \mathbb{R}$ برای مسئله‌ی زیر وجود خواهد داشت [۲]:

$$\max_{\|f\|_L \leq ۱} E_{x \sim P_{data}} [f(x)] - E_{x \sim P_\theta} [f(x)]$$

و رابطه زیر برقرار است:

$$\nabla_\theta \mathbf{W}(P_{data}, P_\theta) = -E_{z \sim P(z)} [\nabla_\theta f(g_\theta(z))]$$

قضیه ۳-۱-۸. دنباله‌ای از توزیع $(P_t)_{t \in N}$ همگراست اگر و تنها اگر یک توزیع P_∞ وجود داشته باشد به طوری که $\rho(P_t, P_\infty)$ به صفر تمایل داشته باشد [۲].

قضیه ۳-۱-۹ (ناپدید شدن شیب روی شبکه مولد). فرض کنید $g_\theta : \mathcal{Z} \to \mathcal{X}$ یک تابع قابل تشخیص باشد که دارای توزیع P_g است. با در نظر گرفتن P_{data} به عنوان توزیع نمونه‌های واقعی و $\mathbf{D}$ به عنوان شبکه متمایزکننده، یک شبکه متمایزکننده بهینه هموار $\mathbf{D}' : \mathcal{X} \to [۰, ۱]$ با دقت ۱ وجود دارد به صورتی که برای تمام x های درون منیفولد، رابطه $\nabla_x \mathbf{D}'(x) = ۰$ برقرار باشد. در این صورت اگر $\|\mathbf{D} - \mathbf{D}'\| < \epsilon$ و $E_{z \sim P(z)} \|\mathbf{J}_\theta g_\theta(z) \leq M^۲$ برقرار باشد سپس

$$\| \nabla_\theta E_{z \sim P(z)} \log (۱ - \mathbf{D}(g_\theta(z))) \|_۲ < M \frac{\epsilon}{۱ - \epsilon}$$

اثبات این قضیه در مرجع [۴۴] قرار دارد.

۳-۲ ساختار شبکه مولد رقابتی وَسِراِستین

شبکه مولد رقابتی وَسِراِستین [۱] نوع توسعه یافته از شبکه مولد رقابتی (بخش ۲-۳) است که ساختاری تقریبا مشابه با شبکه GAN دارد. شبکه مولد رقابتی وَسِراِستین یا همان WGAN از ترکیب دو شبکه عصبی مولد و منتقد [۲] استفاده می‌کند و نقش شبکه متمایزکننده را ضعیف می‌کند. در فصل قبل (بخش ۲-۳-۱) دیدید که شبکه متمایزکننده در شبکه مولد رقابتی با داشتن تابع فعالیت همچون sigmoid در لایه‌ی آخر خود، طبقه‌بندی نمونه‌های واقعی و نمونه‌های تولید شده را انجام می‌دهد. این شبکه با تابع هزینه بیان شده، آموزش می‌بیند و میزان

[۱] Wasserstein Generative Adversial Network (WGAN) [۲] Critic

اختلاف خروجی تولید شده توسط شبکه مولد و خروجی واقعی نمونه‌ها را به عنوان خطا محاسبه می‌کند. این میزان خطا با قاعده پس انتشار خطا به عقب بازگشته و پارامترهای شبکه را به‌هنگام می‌کنند، از این رو شبکه مولد و شبکه متمایزکننده در مراحل بعدی بهتر عمل می‌کنند.

شبکه عصبی مولد در شبکه WGAN همانند شبکه مولد در شبکه GAN هاست که در بخش ۲-۳-۲ مورد بررسی قرار گرفت. شبکه منتقد در شبکه WGAN یک شبکه عصبی با لایه‌های متفاوت است. روش خاص برای چیدن لایه‌ها در کنار یکدیگر وجود ندارد و از لحاظ تجربی به موارد قابل قبولی دست یافتند. ورودی شبکه منتقد تنسور با ابعاد بالا هستند؛ به عنوان مثال تصاویر $۳ \times ۲۸ \times ۲۸$ که تنسور سه بعدی هستند و در بخش ۱-۳ بیان شدند. شبکه منتقد به عنوان خروجی برای پیش‌بینی، یک نمره واقعیت [۱] برای هر نمونه‌ی معین در نظر می‌گیرد. از این رو باید ابعاد ورودی اولیه کاهش یابد. در این شبکه اکثراً از لایه‌هایی همچون لایه‌ی مسطح سازی (بخش ۱-۴-۳) و لایه‌های پیچشی با گام حرکت بالا برای پنجره لغزان استفاده می‌شود که ابعاد و تعداد نورون‌های هر لایه را کاهش دهد. در لایه‌ی آخر از یک تابع فعالیت خطی به‌جای تابع sigmoid استفاده می‌شود تا یک نمره یا امتیاز واقعیت به هر نمونه اختصاص دهد [۴۵].

۳-۳ آموزش شبکه مولد رقابتی وَسِراِستین

در شبکه مولد رقابتی GAN که در فصل قبل به آن پرداخته شد، دو شبکه عصبی مولد و متمایزکننده در یک بازی مجموع-صفر به صورت پی در پی آموزش می‌بینند. این آموزش تا رسیدن به نقطه تعادل نش ادامه می‌یابد. شبکه WGAN در هنگام آموزش نقش شبکه متمایزکننده را ضعیف کرده و از یک شبکه منتقد استفاده می‌کند که به جای طبقه‌بندی یا پیش‌بینی، احتمال واقعی بودن یک نمونه را مشخص می‌کند. به عنوان تابع هزینه در آموزش شبکه WGAN، یک تقریب کاربردی از فاصله‌ی وَسِراِستین $\mathbf{W}$ یا EMD (بخش ۳-۱-۱) بکار برده می‌شود. طبق قضیه ۳-۱-۶ فاصله $\mathbf{W}(P_{data}, P_\theta)$ در هنگام بهینه سازی از واگرایی جیسون-شانون $JS(P_{data}, P_\theta)$ عملکرد بهتری دارد. اما به دست آوردن اینفیمم (inf) در رابطه فاصله‌ی وَسِراِستین (۳-۴) به آسانی امکان‌پذیر نیست. از این رو با استفاده از دوگانگی کانتورویچ-روبنشتاین [۲] رابطه به صورت زیر بازنویسی می‌شود [۲].

$$\mathbf{W}(P_{data}, P_\theta) = \sup_{\|f\|_L \leq ۱} E_{x \sim P_{data}}[f(x)] - E_{x \sim P_\theta}[f(x)] \qquad (۳-۵)$$

[۱] Realness Score [۲] Kantorovich-Rubinstein Duality

در رابطه بالا مقدار سوپریمم (sup) بر روی تمام توابع لیپشیتز با ثابت ۱ ($f : \mathcal{X} \to R$) است. اگر حد آستانه $\|f\|_L \leq ۱$ با مقدار $\|f\|_L \leq K$ جایگزین شود آنگاه فاصله در مقدار ثابت K، برابر $K \cdot \mathbf{W}(P_{data}, P_\theta)$ خواهد بود. بنابراین اگر خانواده‌ای از توابع لیپشیتز با ثابت K همانند $\{f_w\}_{w \in \mathrm{w}}$ داشته باشیم، مسئله به صورت زیر حل می‌شود [۲]:

$$\max_{w \in \mathrm{w}} E_{x \sim P_{data}} [f_w(x)] - E_{z \sim P_z} [f_w(g_\theta(z))] \qquad (۶-۳)$$

اگر sup در معادله (۳-۵) برای برخی از $w \in W$ به‌دست آید، مقدار $\mathbf{W}(P_{data}, P_\theta)$ در مقدار ثابت K ضرب می‌شود. علاوه بر این مشتق $\mathbf{W}(P_{data}, P_\theta)$ توسط قاعده پس‌انتشار رابطه (۳-۵) با تخمین میزان رابطه زیر $E_{z \sim P_z} [\nabla_\theta f_w(g_\theta(z))]$ به دست می‌آید. بهینه بودن این موارد در قضیه ۳-۱-۷ بیان شده است. برای به دست آوردن تابع f برای حل مسئله بیشینه‌سازی (۳-۵) می‌توان پارامترهای یک شبکه عصبی با وزن‌های $w \in W$ را آموزش داده و سپس توسط $E_{z \sim P_z} [\nabla_\theta f_w(g_\theta(z))]$ پس‌انتشار به عقب همانند شبکه مولد رقابتی معمولی انجام شود. تمام توابع f به صورت توابع لیپشیتز با ثابت K هستند که به W وابسته اند و به وزن‌های تکی w بستگی ندارند. از این رو W باید پیوسته و متراکم باشد. برای این منظور پس از هر به‌هنگام سازی شیب گرادیان، وزن‌ها در یک محیط ثابت (مثلا $W = [-۰/۰۱, ۰/۰۱]^L$) قرار داده می‌شوند [۲].

شبکه منتقد بر طبق رابطه (۳-۶) فاصله‌ی وَسِراِستین میان توزیع نمونه‌های تولید شده توسط شبکه مولد و توزیع نمونه‌های واقعی را محاسبه می‌کند و در تلاش است که این فاصله بیشینه شود. با بیشتر شدن فاصله‌ی توزیع نمونه‌های تولید شده توسط مولد و توزیع نمونه‌های واقعی، شبکه منتقد به خوبی نمره‌ای به نمونه‌ها می‌دهد. از طرف دیگر شبکه مولد برای تولید نمونه‌های مناسب که تقریباً به نمونه‌های واقعی مشابه هستند، سعی در کمینه کردن فاصله‌ی توزیع نمونه‌های تولید شده و توزیع نمونه‌های واقعی دارد. از این رو تابع هزینه شبکه مولد به صورت زیر خواهد بود [۲۲]:

$$\min_{w \in \mathrm{w}} E_{x \sim P_{data}} [f_w(x)] - E_{z \sim P_z} [f_w(g_\theta(z))]$$

در آموزش این بخش فقط از نمونه‌های تولید شده توسط شبکه مولد استفاده می‌شود، از این رو رابطه بالا به صورت زیر بازنویسی می‌شود:

$$\min_{w \in \mathrm{w}} -E_{z \sim P_z} [f_w(g_\theta(z))] \qquad (۷-۳)$$

به عبارتی تابع هزینه شبکه منتقد روی کیفیت تصاویر ایجاد شده توسط شبکه مولد تأثیر دارد؛ هرچه مقدار تابع هزینه شبکه منتقد در هنگام ارزیابی نمونه‌های تولید شده توسط شبکه مولد کمتر باشد، کیفیت مورد انتظار نمونه‌های تولیدی بالاتر می‌رود. از این رو کمینه کردن تابع هزینه‌ی شبکه مولد از اهمیت برخوردار است. به همین منظور تابع هزینه شبکه WGAN به صورت کمینه سازی رابطه زیر خواهد بود:

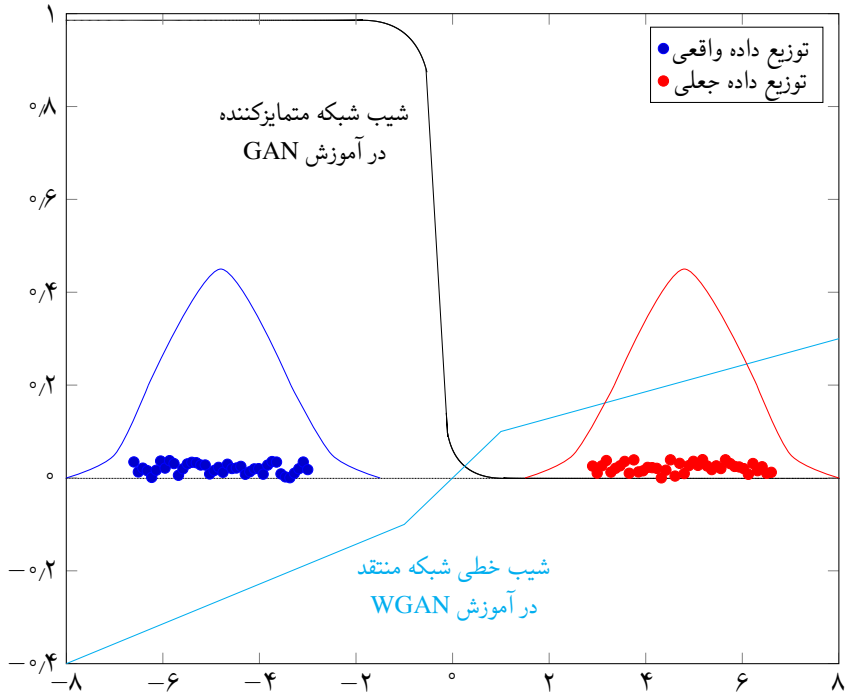
$$\min_{w \in \mathrm{W}} E_{x \sim P_{data}} [f_w(x)] - E_{z \sim P_z} [f_w(g_\theta(z))] \qquad (۸-۳)$$

برای کمینه کردن این رابطه شبکه مولد در تلاش است که شبکه منتقد نمره یا امتیاز بالایی برای نمونه‌های تولید شده توسط شبکه مولد، اختصاص دهد. گرفتن امتیاز بالا از شبکه منتقد برای نمونه‌های تولید شده منجر به کمینه سازی تابع هزینه شبکه مولد می‌شود. از طرف دیگر شبکه منتقد برای بیشینه کردن تابع هزینه‌ی خود (رابطه (۳-۶)) امتیاز بالاتری به نمونه‌های واقعی می‌دهد اما به دلیل آنکه تابع هزینه بالا باید کمینه شود، شبکه منتقد مجبور به اختصاص امتیاز کم به نمونه‌های واقعی می‌شود [۴۶].

پیوستگی و متفاوت بودن فاصله‌ی وَسِراِستین $\mathbf{W}$ در مکان‌های مختلف به این معناست که می‌توان شبکه منتقد را تا زمان بهینه شدن آموزش داد؛ شبکه منقد هرچه بیشتر آموزش داده شود، به شیب معتبرتری از وَسِراِستین می‌رسد. برای JS هرچه شبکه متمایزکننده بهتر باشد شیب معتبری ارائه می‌دهد اما شیب واقعی صفر است زیرا پس از تعداد معینی آموزش، این شبکه اشباع شده و دچار بیش برازش [۱] می‌شود. بدین معنی که شبکه روی داده‌های آموزشی قوی شده و به خوبی جواب می‌دهد ولی روی داده‌های جدید عملکرد خوبی ندارد. قضیه ۳-۱-۹ به این موضوع اشاره دارد. در شکل ۳-۲ یک شبکه عصبی متمایزکننده در شبکه مولد رقابتی GAN و یک شبکه عصبی منتقد در شبکه WGAN آموزش داده شده است. شبکه متمایزکننده خیلی سریع تمایز بین نمونه‌های جعلی و واقعی را یاد می‌گیرد اما شیب مطلوبی را ارائه نمی‌دهد. این شبکه پس از مدتی دچار بیش برازش شده و فقط روی نمونه‌های آموزشی به درستی کار می‌کند و برای نمونه‌های تولیدشده عملکرد خوبی ندارد. شبکه منتقد اشباع نمی‌شود و به یک عملکرد خطی تبدیل می‌شود که دارای شیب مناسبی در همه‌ی مکان هاست. این شبکه با آموزش بیشتر، دچار بیش برازش نمی‌شود بلکه عملکرد بهتری روی داده‌های تولیدشده دارد. دلیل این کار محدود کردن وزن‌هاست زیرا رشد تابع را در قسمت‌های مختلف به صورت خطی می‌کند. به همین دلیل در شکل هیچ سقوطی برای شبکه منتقد وجود ندارد و با آموزش بیشتر عملکرد این شبکه بهتر خواهد بود [۲].

تابع هزینه در شبکه‌های WGAN خواص همگرایی دارد. این ویژگی در تحقیقات روی شبکه‌های رقابتی بسیار مفید است؛ زیرا نیاز به چک کردن نمونه‌های خروجی برای جدا کردن نمونه‌های خراب‌شده نیست. از طرفی اطلاعاتی در خصوص عملکرد بهتر این شبکه‌ها نسبت به بقیه به دست خواهد آمد.

[۱] overfitting



شکل ۳-۲: عملکرد شبکه متمایزکننده و شبکه منتقد روی نمونه‌های واقعی و تولید شده. شبکه متمایزکننده در GAN اشباع شده و دچار overffiting می‌شود اما شبکه منتقد در WGAN در همه قسمت‌ها شیب مناسبی دارد.

الگوریتم ۳-۱ الگوریتم گرادیان تابع هزینه وَسِراِستین [۴۷]

ورودی:

خروجی: و

۱: مراحل زیر را تا زمانی که u همگرا نشده است، ادامه دهید.

۳-۴ الگوریتم شبکه مولد رقابتی وَسِراِستین

چارچوب کلی شبکه WGAN در الگوریتم ۳-۲ آمده است. در ابتدا شبکه منتقد برای تعداد دفعات مشخص آموزش داده می‌شود. برای این منظور در هر تکرار از آموزش شبکه منتقد، تعداد مشخصی از نمونه‌های واقعی و نمونه‌های ساخته شده به شبکه منتقد داده و وزن‌های این شبکه به‌هنگام می‌شود. پس از هر به‌هنگام سازی دسته‌ای از وزن‌های مدل منتقد، این وزن‌ها در یک بازه‌ی خاص مانند $[-۰/۰۱, ۰/۰۱]$ محدود می‌شوند. نکته قابل توجه این است که به‌هنگام سازی وزن‌های شبکه منتقد برخلاف شبکه‌های مولد و متمایزکننده، در آموزش خود شبکه انجام می‌شود و در آموزش شبکه WGAN فقط به‌هنگام سازی وزن‌های مولد صورت می‌گیرد. شبکه منتقد برای نمونه‌های واقعی و جعلی، امتیازهای خاصی در نظر می‌گیرد به صورتی که بتواند واقعی یا جعلی بودن نمونه‌ها را تشخیص دهد. به عنوان مثال برای نمونه‌های واقعی امتیاز بالا و برای نمونه‌های ساخته شده امتیاز کم در نظر می‌گیرد. پس از چندین مرتبه آموزش شبکه منتقد، این شبکه متوقف شده و شبکه مولد برای یکبار آموزش میبیند. شبکه مولد یک بردار اولیه تصادفی را به عنوان ورودی دریافت می‌کند و نمونه‌ی مورد نظر (مثلاً یک تصویر) را تولید می‌کند. شبکه مولد در تلاش است که کیفیت نمونه‌های تولیدی را همانند نمونه‌های واقعی کند و امتیاز بالایی از شبکه منتقد برای نمونه‌های تولیدی بگیرد. پس از آن مجدداً شبکه منتقد برای تعداد دفعات مشخص آموزش می‌بینند. این روند تا تولید نمونه‌های با کیفیت ادامه می‌یابد [۴۶].

برای پیاده سازی شبکه WGAN به‌جای محاسبه‌ی تابع هزینه شبکه منتقد، می‌توان از امتیاز یا نمره منفی برای نمونه‌های واقعی و یک امتیاز مثبت برای نمونه‌های جعلی استفاده نمود. بدین صورت که امتیازی که شبکه منتقد برای نمونه‌های واقعی در نظر می‌گیرد، در عدد ۱ و امتیاز نمونه‌های جعلی در -۱ ضرب می‌شود. در این صورت تابع هزینه به صورت ضرب برچسب‌های ۱ و -۱ در امتیاز شبکه منتقد به دست می‌آید. برنامه‌ی آ-۸ شبکه مولد رقابتی وَسِراِستین برای تولید یک رقم از مجموعه ارقام دست‌نویس MNIST را بیان کرده است. خروجی این برنامه برای دو عدد ۶ و ۷ در شکل ۳-۳ آمده است [۴۵].

(آ) تصاویر ساخته شده از عدد۶ (ب) تصاویر ساخته شده از عدد ۷

شکل ۳-۳: تصاویر تولید شده توسط WGAN با آموزش روی مجموعه داده MNIST

الگوریتم ۳-۲ الگوریتم آموزش شبکه WGAN [۲]

ورودی: نرخ خطای α، پارامتر قطع کردن c، اندازه دسته‌ها (m)،
تعداد تکرار شبکه منتقد در هر تکرار شبکه مولد (n_{critic})،
وزن های اولیه شبکه منتقد $(w_۰)$ و ورن های اولیه شبکه مولد $(\theta_۰)$
خروجی: تولید تصاویر جدید

۱: عملیات زیر را تا زمانی که θ همگرا نشده است، انجام دهید.
۲: مراحل زیر را به تعداد $t = ۰, \ldots, n_{critic}$ انجام دهید.
- انتخاب یک دسته‌ی mتایی از داده‌های واقعی $\{x_i\}_{i=۱}^{m} \sim P_{data}$
- انتخاب یک دسته‌ی mتایی از بردار اولیه تصادفی $\{z_i\}_{i=۱}^{m} \sim P_z$
- محاسبه رابطه زیر

$$g_w = \nabla_w \left[\frac{۱}{m} \sum_{i=۱}^{m} f_w(x_i) - \frac{۱}{m} \sum_{i=۱}^{m} f_w(g_\theta(z_i)) \right]$$

- محاسبه میزان وزن شبکه مولد به صورت زیر

$$w_{new} = wold + \alpha \cdot RMSProp(w, g_w)$$

- محاسبه میزان وزن

$$w_{new} = clip(w, -c, c)$$

۳: انتخاب یک دسته‌ی mتایی از بردار اولیه تصادفی $\{z_i\}_{i=۱}^{m} \sim P_z$
۴: محاسبه رابطه زیر

$$g_\theta = -\nabla_\theta \frac{۱}{m} \sum_{i=۱}^{m} f_w(g_\theta(z_i))$$

۵: محاسبه‌ی وزن‌های شبکه منتقد

$$\theta_{new} = \theta old - \alpha \cdot RMSProp(\theta, g_\theta)$$

فهرست منابع

[1] Vasilev, I., Slater, D., Spacagna, G., Roelants, P., and Zocca, V. *Python Deep Learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow, 2nd Edition*. Packt Publishing, 2019.

[2] Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein generative adversarial networks. in Precup, Doina and Teh, Yee Whye, eds. , *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, pp. 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[۳] منهاج، محمد باقر. هوش محاسباتی - جلد اول: مبانی شبکه‌های عصبی. دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)، ۱۳۹۳.

[4] Chollet, François. *Deep Learning with Python*. Manning, November 2017.

[5] Sibi, P, Jones, S Allwyn, and Siddarth, P. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology*, 47(3):1264–1268, 2013.

[6] Elfwing, Stefan, Uchibe, Eiji, and Doya, Kenji. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[7] Nie, Feiping, Zhanxuan, Hu, and Li, Xuelong. An investigation for loss functions widely used in machine learning. *Communications in Information and Systems*, 18:37–52, 01 2018.

[8] Qian, Ning. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[9] Läthén, Gunnar, Andersson, Thord, Lenz, Reiner, and Borga, Magnus. Momentum based optimization methods for level set segmentation. in *International Conference*

on Scale Space and Variational Methods in Computer Vision, pp. 124–136. Springer, 2009.

[10] Aggarwal, Charu C. *Neural Networks and Deep Learning*. Springer, Cham, 2018.

[11] Dozat, Timothy. Incorporating nesterov momentum into adam. 2016.

[12] Zhang, Z. Improved adam optimizer for deep neural networks. in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–2, June 2018.

[13] Dangeti, Pratap. *Statistics for machine learning*. Packt Publishing Ltd, 2017.

[۱۴] جکسون، راسل بیل و تام. آشنایی با شبکه‌های عصبی. ترجمه‌ی البرزی، محمود. دانشگاه صنعتی شریف، موسسه انتشارات علمی، ۱۳۹۳.

[15] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[16] Gu, Jiuxiang, Wang, Zhenhua, Kuen, Jason, Ma, Lianyang, Shahroudy, Amir, Shuai, Bing, Liu, Ting, Wang, Xingxing, Wang, Gang, Cai, Jianfei, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[17] Brownlee, Jason. Machine learning mastery, how to use the upsampling2d and conv2dtranspose layers in keras. https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks, 2019.

[18] Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] Foster, D. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media, 2019.

[21] Karras, Tero, Aila, Timo, Laine, Samuli, and Lehtinen, Jaakko. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[22] Langr, J. and Bok, V. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications, 2019.

[23] Myerson, R.B. *Game Theory: Analysis of Conflict*. Harvard University Press, 2013.

[24] Osborne, Martin J et al. *An introduction to game theory*, vol. 3. Oxford university press New York, 2004.

[25] Prisner, Erich. *Game Theory Through Examples*. 01 2014.

[26] Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. in Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., eds. , *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014.

[27] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. in Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., eds. , *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[28] Brownlee, Jason. Machine learning mastery, how to develop a gan for generating mnist handwritten digits. https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/, 2019.

[29] Bruce, Peter and Bruce, Andrew. *Practical statistics for data scientists: 50 essential concepts*. " O'Reilly Media, Inc.", 2017.

[30] Brownlee, Jason. Machine learning mastery, how to develop a 1d generative adversarial network from scratch in keras. https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-1-dimensional-function-from-scratch-in-keras/, 2019.

[31] Arjovsky, Martin and Bottou, Léon. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[32] Bissoto, Alceu, Valle, Eduardo, and Avila, Sandra. The six fronts of the generative adversarial networks. *arXiv preprint arXiv:1910.13076*, 2019.

[33] Mirza, Mehdi and Osindero, Simon. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[34] Denton, Emily L, Chintala, Soumith, Fergus, Rob, et al. Deep generative image models using a laplacian pyramid of adversarial networks. in *Advances in neural information processing systems*, pp. 1486–1494, 2015.

[35] Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[36] Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[37] Odena, Augustus, Olah, Christopher, and Shlens, Jonathon. Conditional image synthesis with auxiliary classifier gans. in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651. JMLR. org, 2017.

[38] Zhu, Jun-Yan, Park, Taesung, Isola, Phillip, and Efros, Alexei A. Unpaired image-to-image translation using cycle-consistent adversarial networks. in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[39] Mo, Sangwoo, Cho, Minsu, and Shin, Jinwoo. Instagan: Instance-aware image-to-image translation. *arXiv preprint arXiv:1812.10889*, 2018.

[40] Brock, Andrew, Donahue, Jeff, and Simonyan, Karen. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[41] Karras, Tero, Laine, Samuli, and Aila, Timo. A style-based generator architecture for generative adversarial networks. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

[42] Nowozin, Sebastian, Cseke, Botond, and Tomioka, Ryota. f-gan: Training generative neural samplers using variational divergence minimization. in *Advances in neural information processing systems*, pp. 271–279, 2016.

[43] Huszár, Ferenc. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*, 2015.

[44] Arjovsky, Martín and Bottou, Léon. Towards principled methods for training generative adversarial networks. *ArXiv*, abs/1701.04862, 2017.

[45] Brownlee, Jason. Machine learning mastery, how to develop a wasserstein generative adversarial network (wgan) from scratch. https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch, 2019.

[46] Brownlee, Jason. Machine learning mastery, how to implement wasserstein loss for generative adversarial networks. https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks, 2019.

[47] Frogner, Charlie, Zhang, Chiyuan, Mobahi, Hossein, Araya, Mauricio, and Poggio, Tomaso A. Learning with a wasserstein loss. in *Advances in Neural Information Processing Systems*, pp. 2053–2061, 2015.

پیوست آ

برنامه‌های مرتبط با شبکه مولد رقابتی

برنامه آ-۱: برنامه پایتون قاعده پس انتشار خطا

برنامه آ-۲: مدل شبکه متمایزکننده

```
# example of defining the discriminator model
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LeakyReLU
from keras.utils.vis_utils import plot_model

# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,3)):
        model = Sequential()
        model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',
            input_shape=in_shape))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.4))
        model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.4))
        model.add(Flatten())
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt,
            metrics=['accuracy'])
        return model

# define model
```

```
model = define_discriminator()
# summarize the model
model.summary()
# plot the model
plot_model(model, to_file='discriminator_plot.png', show_shapes=True,
    show_layer_names=True)
```

برنامه آ-۳: مدل شبکه مولد

```
# example of defining the generator model
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.utils.vis_utils import plot_model

# define the standalone generator model
def define_generator(latent_dim):
        model = Sequential()
        # foundation for 7x7 image
        n_nodes = 128 * 7 * 7
        model.add(Dense(n_nodes, input_dim=latent_dim))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Reshape((7, 7, 128)))
        # upsample to 14x14
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='
            same'))
        model.add(LeakyReLU(alpha=0.2))
        # upsample to 28x28
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='
            same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2D(3, (7,7), activation='sigmoid', padding='same'
            ))
        return model

# define the size of the latent space
latent_dim = 100
# define the generator model
model = define_generator(latent_dim)
# summarize the model
model.summary()
# plot the model
plot_model(model, to_file='generator_plot.png', show_shapes=True,
    show_layer_names=True)
```

برنامه آ-۴: آموزش شبکه عصبی متمایزکننده به تنهایی

```
# define and fit a discriminator model
from numpy import zeros
```

```
from numpy import ones
from numpy import hstack
from numpy.random import rand
from numpy.random import randn
from keras.models import Sequential
from keras.layers import Dense

# define the standalone discriminator model
def define_discriminator(n_inputs=2):
        model = Sequential()
        model.add(Dense(25, activation='relu', kernel_initializer='
            he_uniform', input_dim=n_inputs))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        model.compile(loss='binary_crossentropy', optimizer='adam',
            metrics=['accuracy'])
        return model

# generate n real samples with class labels
def generate_real_samples(n):
        # generate inputs in [-0.5, 0.5]
        X1 = rand(n) - 0.5
        # generate outputs X^2
        X2 = X1 * X1
        # stack arrays
        X1 = X1.reshape(n, 1)
        X2 = X2.reshape(n, 1)
        X = hstack((X1, X2))
        # generate class labels
        y = ones((n, 1))
        return X, y

# generate n fake samples with class labels
def generate_fake_samples(n):
        # generate inputs in [-1, 1]
        X1 = -1 + rand(n) * 2
        # generate outputs in [-1, 1]
        X2 = -1 + rand(n) * 2
        # stack arrays
        X1 = X1.reshape(n, 1)
        X2 = X2.reshape(n, 1)
        X = hstack((X1, X2))
        # generate class labels
        y = zeros((n, 1))
        return X, y

# train the discriminator model
def train_discriminator(model, n_epochs=1000, n_batch=128):
        half_batch = int(n_batch / 2)
        # run epochs manually
        for i in range(n_epochs):
                # generate real examples
                X_real, y_real = generate_real_samples(half_batch)
                # update model
                model.train_on_batch(X_real, y_real)
                # generate fake examples
```

```
                X_fake, y_fake = generate_fake_samples(half_batch)
                # update model
                model.train_on_batch(X_fake, y_fake)
                # evaluate the model
                _, acc_real = model.evaluate(X_real, y_real, verbose=0)
                _, acc_fake = model.evaluate(X_fake, y_fake, verbose=0)
                print(i,'     ','acc_real = ', acc_real, '    ','
                    acc_fake = ', acc_fake)

# define the discriminator model
model = define_discriminator()
# fit the model
train_discriminator(model)
```

برنامه آ-۵: آموزش شبکه عصبی مولد به تنهایی

```
# define and use the generator model
from numpy.random import randn
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot

# define the standalone generator model
def define_generator(latent_dim, n_outputs=2):
        model = Sequential()
        model.add(Dense(15, activation='relu', kernel_initializer='
            he_uniform', input_dim=latent_dim))
        model.add(Dense(n_outputs, activation='linear'))
        return model

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n):
        # generate points in the latent space
        x_input = randn(latent_dim * n)
        # reshape into a batch of inputs for the network
        x_input = x_input.reshape(n, latent_dim)
        return x_input

# use the generator to generate n fake examples and plot the results
def generate_fake_samples(generator, latent_dim, n):
        # generate points in latent space
        x_input = generate_latent_points(latent_dim, n)
        # predict outputs
        X = generator.predict(x_input)
        # plot the results
        pyplot.scatter(X[:, 0], X[:, 1])
        pyplot.show()

# size of the latent space
latent_dim = 5
# define the discriminator model
model = define_generator(latent_dim)
# generate and plot generated samples
generate_fake_samples(model, latent_dim, 100)
```

برنامه آ-۶: برنامه پایتون شبکه مولد رقابتی برای تولید نقاط تابع x^2

```
# train a generative adversarial network on a one-dimensional function
from numpy import hstack
from numpy import zeros
from numpy import ones
from numpy.random import rand
from numpy.random import randn
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot

# define the standalone discriminator model
def define_discriminator(n_inputs=2):
        model = Sequential()
        model.add(Dense(25, activation='relu', kernel_initializer='
            he_uniform', input_dim=n_inputs))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        model.compile(loss='binary_crossentropy', optimizer='adam',
            metrics=['accuracy'])
        return model

# define the standalone generator model
def define_generator(latent_dim, n_outputs=2):
        model = Sequential()
        model.add(Dense(15, activation='relu', kernel_initializer='
            he_uniform', input_dim=latent_dim))
        model.add(Dense(n_outputs, activation='linear'))
        return model

# define the combined generator and discriminator model, for updating
    the generator
def define_gan(generator, discriminator):
        # make weights in the discriminator not trainable
        discriminator.trainable = False
        # connect them
        model = Sequential()
        # add generator
        model.add(generator)
        # add the discriminator
        model.add(discriminator)
        # compile model
        model.compile(loss='binary_crossentropy', optimizer='adam')
        return model

# generate n real samples with class labels
def generate_real_samples(n):
        # generate inputs in [-0.5, 0.5]
        X1 = rand(n) - 0.5
        # generate outputs X^2
        X2 = X1 * X1
```

```
        # stack arrays
        X1 = X1.reshape(n, 1)
        X2 = X2.reshape(n, 1)
        X = hstack((X1, X2))
        # generate class labels
        y = ones((n, 1))
        return X, y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n):
        # generate points in the latent space
        x_input = randn(latent_dim * n)
        # reshape into a batch of inputs for the network
        x_input = x_input.reshape(n, latent_dim)
        return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n):
        # generate points in latent space
        x_input = generate_latent_points(latent_dim, n)
        # predict outputs
        X = generator.predict(x_input)
        # create class labels
        y = zeros((n, 1))
        return X, y

# evaluate the discriminator and plot real and fake points
def summarize_performance(epoch, generator, discriminator, latent_dim,
    n=100):
        # prepare real samples
        x_real, y_real = generate_real_samples(n)
        # evaluate discriminator on real examples
        _, acc_real = discriminator.evaluate(x_real, y_real, verbose=0)
        # prepare fake examples
        x_fake, y_fake = generate_fake_samples(generator, latent_dim, n
            )
        # evaluate discriminator on fake examples
        _, acc_fake = discriminator.evaluate(x_fake, y_fake, verbose=0)
        # summarize discriminator performance
        print(epoch, acc_real, acc_fake)
        # scatter plot real and fake data points
        pyplot.scatter(x_real[:, 0], x_real[:, 1], color='red')
        pyplot.scatter(x_fake[:, 0], x_fake[:, 1], color='blue')
        pyplot.show()

# train the generator and discriminator
def train(g_model, d_model, gan_model, latent_dim, n_epochs=10000,
    n_batch=128, n_eval=2000):
        # determine half the size of one batch, for updating the
            discriminator
        half_batch = int(n_batch / 2)
        # manually enumerate epochs
        for i in range(n_epochs):
                # prepare real samples
                x_real, y_real = generate_real_samples(half_batch)
                # prepare fake examples
```

```
                x_fake, y_fake = generate_fake_samples(g_model,
                    latent_dim, half_batch)
                # update discriminator
                d_model.train_on_batch(x_real, y_real)
                d_model.train_on_batch(x_fake, y_fake)
                # prepare points in latent space as input for the
                    generator
                x_gan = generate_latent_points(latent_dim, n_batch)
                # create inverted labels for the fake samples
                y_gan = ones((n_batch, 1))
                # update the generator via the discriminator's error
                gan_model.train_on_batch(x_gan, y_gan)
                # evaluate the model every n_eval epochs
                if (i+1) % n_eval == 0:
                        summarize_performance(i, g_model, d_model,
                            latent_dim)

# size of the latent space
latent_dim = 5
# create the discriminator
discriminator = define_discriminator()
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# train model
train(generator, discriminator, gan_model, latent_dim)
```

برنامه آ-۷: برنامه پایتون شبکه مولد رقابتی برای تولید تصاویر MNIST

```
# example of training a gan on mnist
from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy import vstack
from numpy.random import randn
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Dropout
from matplotlib import pyplot
import matplotlib.pyplot as plt

# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
        model = Sequential()
        model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',
```

```
            input_shape=in_shape))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.4))
        model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.4))
        model.add(Flatten())
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt,
            metrics=['accuracy'])
        return model

# define the standalone generator model
def define_generator(latent_dim):
        model = Sequential()
        # foundation for 7x7 image
        n_nodes = 128 * 7 * 7
        model.add(Dense(n_nodes, input_dim=latent_dim))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Reshape((7, 7, 128)))
        # upsample to 14x14
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='
            same'))
        model.add(LeakyReLU(alpha=0.2))
        # upsample to 28x28
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='
            same'))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Conv2D(1, (7,7), activation='sigmoid', padding='same'
            ))
        return model

# define the combined generator and discriminator model, for updating
    the generator
def define_gan(g_model, d_model):
        # make weights in the discriminator not trainable
        d_model.trainable = False
        # connect them
        model = Sequential()
        # add generator
        model.add(g_model)
        # add the discriminator
        model.add(d_model)
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt)
        return model

# load and prepare mnist training images
def load_real_samples():
        # load mnist dataset
        (trainX, _), (_, _) = load_data()
        # expand to 3d, e.g. add channels dimension
        X = expand_dims(trainX, axis=-1)
```

```
        # convert from unsigned ints to floats
        X = X.astype('float32')
        # scale from [0,255] to [0,1]
        X = X / 255.0
        return X

# select real samples
def generate_real_samples(dataset, n_samples):
        # choose random instances
        ix = randint(0, dataset.shape[0], n_samples)
        # retrieve selected images
        X = dataset[ix]
        # generate 'real' class labels (1)
        y = ones((n_samples, 1))
        return X, y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
        # generate points in the latent space
        x_input = randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
        x_input = x_input.reshape(n_samples, latent_dim)
        return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
        # generate points in latent space
        x_input = generate_latent_points(latent_dim, n_samples)
        # predict outputs
        X = g_model.predict(x_input)
        # create 'fake' class labels (0)
        y = zeros((n_samples, 1))
        return X, y

# create and save a plot of generated images (reversed grayscale)
def save_plot(examples, epoch, n=10):
        # plot images
        for i in range(n * n):
                # define subplot
                pyplot.subplot(n, n, 1 + i)
                # turn off axis
                pyplot.axis('off')
                # plot raw pixel data
                pyplot.imshow(examples[i, :, :, 0], cmap='gray_r')
        # save plot to file
        filename = 'generated_plot_e%03d.png' % (epoch+1)
        pyplot.savefig(filename)
        pyplot.show()
        pyplot.close()

# evaluate the discriminator, plot generated images, save generator
    model
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim,
    n_samples=100):
        # prepare real samples
        X_real, y_real = generate_real_samples(dataset, n_samples)
```

```
        # evaluate discriminator on real examples
        _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
        # prepare fake examples
        x_fake, y_fake = generate_fake_samples(g_model, latent_dim,
            n_samples)
        # evaluate discriminator on fake examples
        _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
        # summarize discriminator performance
        print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100,
            acc_fake*100))
        # save plot
        save_plot(x_fake, epoch)
        # save the generator model tile file
        filename = 'generator_model_%03d.h5' % (epoch + 1)
        g_model.save(filename)

# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs
    =100, n_batch=256):
        bat_per_epo = int(dataset.shape[0] / n_batch)
        half_batch = int(n_batch / 2)
        # manually enumerate epochs
        for i in range(n_epochs):
                # enumerate batches over the training set
                for j in range(bat_per_epo):
                        # get randomly selected 'real' samples
                        X_real, y_real = generate_real_samples(dataset,
                            half_batch)
                        # generate 'fake' examples
                        X_fake, y_fake = generate_fake_samples(g_model,
                            latent_dim, half_batch)
                        # create training set for the discriminator
                        X, y = vstack((X_real, X_fake)), vstack((y_real
                            , y_fake))
                        # update discriminator model weights
                        d_loss, _ = d_model.train_on_batch(X, y)
                        # prepare points in latent space as input for
                            the generator
                        X_gan = generate_latent_points(latent_dim,
                            n_batch)
                        # create inverted labels for the fake samples
                        y_gan = ones((n_batch, 1))
                        # update the generator via the discriminator's
                            error
                        g_loss = gan_model.train_on_batch(X_gan, y_gan)
                        # summarize loss on this batch
                print('>%d, %d/%d, d=%.3f, g=%.3f' % (i+1, j+1,
                    bat_per_epo, d_loss, g_loss))
                # evaluate the model performance, sometimes
                if (i+1) % 10 == 0:
                        summarize_performance(i, g_model, d_model,
                            dataset, latent_dim)

# size of the latent space
latent_dim = 100
# create the discriminator
```

```
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim)
```

برنامه آ–۸: برنامه پایتون شبکه مولد رقابتی وَسِراستین برای تولید تصویر یک عدد از اعداد MNIST

```
# example of a wgan for generating handwritten digits
from numpy import expand_dims
from numpy import mean
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras import backend
from keras.optimizers import RMSprop
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
from keras.initializers import RandomNormal
from keras.constraints import Constraint
from matplotlib import pyplot

# clip model weights to a given hypercube
class ClipConstraint(Constraint):
        # set clip value when initialized
        def __init__(self, clip_value):
                self.clip_value = clip_value

        # clip model weights to hypercube
        def __call__(self, weights):
                return backend.clip(weights, -self.clip_value, self.
                    clip_value)

        # get the config
        def get_config(self):
                return {'clip_value': self.clip_value}

# calculate wasserstein loss
def wasserstein_loss(y_true, y_pred):
        return backend.mean(y_true * y_pred)

# define the standalone critic model
def define_critic(in_shape=(28,28,1)):
```

```
        # weight initialization
        init = RandomNormal(stddev=0.02)
        # weight constraint
        const = ClipConstraint(0.01)
        # define model
        model = Sequential()
        # downsample to 14x14
        model.add(Conv2D(64, (4,4), strides=(2,2), padding='same',
            kernel_initializer=init, kernel_constraint=const,
            input_shape=in_shape))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.2))
        # downsample to 7x7
        model.add(Conv2D(64, (4,4), strides=(2,2), padding='same',
            kernel_initializer=init, kernel_constraint=const))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.2))
        # scoring, linear activation
        model.add(Flatten())
        model.add(Dense(1))
        # compile model
        opt = RMSprop(lr=0.00005)
        model.compile(loss=wasserstein_loss, optimizer=opt)
        return model

# define the standalone generator model
def define_generator(latent_dim):
        # weight initialization
        init = RandomNormal(stddev=0.02)
        # define model
        model = Sequential()
        # foundation for 7x7 image
        n_nodes = 128 * 7 * 7
        model.add(Dense(n_nodes, kernel_initializer=init, input_dim=
            latent_dim))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Reshape((7, 7, 128)))
        # upsample to 14x14
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='
            same', kernel_initializer=init))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.2))
        # upsample to 28x28
        model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='
            same', kernel_initializer=init))
        model.add(BatchNormalization())
        model.add(LeakyReLU(alpha=0.2))
        # output 28x28x1
        model.add(Conv2D(1, (7,7), activation='tanh', padding='same',
            kernel_initializer=init))
        return model

# define the combined generator and critic model, for updating the
    generator
def define_gan(generator, critic):
        # make weights in the critic not trainable
```

```
        critic.trainable = False
        # connect them
        model = Sequential()
        # add generator
        model.add(generator)
        # add the critic
        model.add(critic)
        # compile model
        opt = RMSprop(lr=0.00005)
        model.compile(loss=wasserstein_loss, optimizer=opt)
        return model

# load images
def load_real_samples():
        # load dataset
        (trainX, trainy), (_, _) = load_data()
        # select all of the examples for a given class
        selected_ix = trainy == 7
        X = trainX[selected_ix]
        # expand to 3d, e.g. add channels
        X = expand_dims(X, axis=-1)
        # convert from ints to floats
        X = X.astype('float32')
        # scale from [0,255] to [-1,1]
        X = (X - 127.5) / 127.5
        return X

# select real samples
def generate_real_samples(dataset, n_samples):
        # choose random instances
        ix = randint(0, dataset.shape[0], n_samples)
        # select images
        X = dataset[ix]
        # generate class labels, -1 for 'real'
        y = -ones((n_samples, 1))
        return X, y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
        # generate points in the latent space
        x_input = randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
        x_input = x_input.reshape(n_samples, latent_dim)
        return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n_samples):
        # generate points in latent space
        x_input = generate_latent_points(latent_dim, n_samples)
        # predict outputs
        X = generator.predict(x_input)
        # create class labels with 1.0 for 'fake'
        y = ones((n_samples, 1))
        return X, y

# generate samples and save as a plot and save the model
```

```
def summarize_performance(step, g_model, latent_dim, n_samples=100):
  # prepare fake examples
  X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
  # scale from [-1,1] to [0,1]
  X = (X + 1) / 2.0
  # plot images
  for i in range(10 * 10):
    # define subplot
    pyplot.subplot(10, 10, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(X[i, :, :, 0], cmap='gray_r')
  # save plot to file
  filename1 = 'generated_plot_%04d.png' % (step+1)
  pyplot.savefig(filename1)
  pyplot.show()
  pyplot.close()
  # save the generator model
  filename2 = 'model_%04d.h5' % (step+1)
  g_model.save(filename2)
  print('>Saved: %s and %s' % (filename1, filename2))

# create a line plot of loss for the gan and save to file
def plot_history(d1_hist, d2_hist, g_hist):
  # plot history
  pyplot.plot(d1_hist, label='crit_real')
  pyplot.plot(d2_hist, label='crit_fake')
  pyplot.plot(g_hist, label='gen')
  pyplot.legend()
  pyplot.savefig('plot_line_plot_loss.png')
  pyplot.show()
  pyplot.close()

# train the generator and critic
def train(g_model, c_model, gan_model, dataset, latent_dim, n_epochs
   =10, n_batch=64, n_critic=5):
        # calculate the number of batches per training epoch
        bat_per_epo = int(dataset.shape[0] / n_batch)
        # calculate the number of training iterations
        n_steps = bat_per_epo * n_epochs
        # calculate the size of half a batch of samples
        half_batch = int(n_batch / 2)
        # lists for keeping track of loss
        c1_hist, c2_hist, g_hist = list(), list(), list()
        # manually enumerate epochs
        for i in range(n_steps):
                # update the critic more than the generator
                c1_tmp, c2_tmp = list(), list()
                for _ in range(n_critic):
                        # get randomly selected 'real' samples
                        X_real, y_real = generate_real_samples(dataset,
                            half_batch)
                        # update critic model weights
                        c_loss1 = c_model.train_on_batch(X_real, y_real
                           )
```

```
                        c1_tmp.append(c_loss1)
                        # generate 'fake' examples
                        X_fake, y_fake = generate_fake_samples(g_model,
                            latent_dim, half_batch)
                        # update critic model weights
                        c_loss2 = c_model.train_on_batch(X_fake, y_fake
                            )
                        c2_tmp.append(c_loss2)
                # store critic loss
                c1_hist.append(mean(c1_tmp))
                c2_hist.append(mean(c2_tmp))
                # prepare points in latent space as input for the
                    generator
                X_gan = generate_latent_points(latent_dim, n_batch)
                # create inverted labels for the fake samples
                y_gan = -ones((n_batch, 1))
                # update the generator via the critic's error
                g_loss = gan_model.train_on_batch(X_gan, y_gan)
                g_hist.append(g_loss)
                # summarize loss on this batch
                print('>%d, c1=%.3f, c2=%.3f g=%.3f' % (i+1, c1_hist
                    [-1], c2_hist[-1], g_loss))
                # evaluate the model performance every 'epoch'
                if (i+1) % bat_per_epo == 0:
                        summarize_performance(i, g_model, latent_dim)
        # line plots of loss
        plot_history(c1_hist, c2_hist, g_hist)

# size of the latent space
latent_dim = 50
# create the critic
critic = define_critic()
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, critic)
# load image data
dataset = load_real_samples()
print(dataset.shape)
# train model
train(generator, critic, gan_model, dataset, latent_dim)
```

واژه‌نامه فارسی به انگلیسی

احتمالی . . . . . Probabilistic

بازی مجموع-صفر . . . . . Zero-Sum

بیش‌برازش . . . . . Overfit

پنجره‌ی لغزان . . . . . Sliding Window

پیوستگی . . . . . Continuity

تابع بهینه سازی . . . . . Optimizer Function

تابع فعالیت(تابع محرک) . . . . . Activation Function

تابع هزینه . . . . . Loss Function

تعادل نش . . . . . Nash Equilibrium

تنسور . . . . . Tensor

خودرمزگذار . . . . . Variational Auto-Encoders(VAE)

داده آموزشی . . . . . Training Data

رویای عمیق . . . . . DeepDream

شبکه پیچشی . . . . . Convolution Network

شبکه عصبی . . . . . Neural Network

شبکه عصبی ایستا یا پیشخور . . . . . Feed-Forward Network

شبکه عصبی بازگشتی یا پسخور . . . . . Feed-Back Network

شبکه متمایزکننده . . . . . Discriminator

شبکه منتقد . . . . . Critic

شبکه مولد . . . . . Generative

شبکه مولد رقابتی (مولد متخاصم) . . . . . Generative Adversaial Network(GAN)

شبکه مولد رقابتی وَسِراِستین (WGAN) Wasserstein Generative Adversial Network

طبقه‌بندی Classification

فاصله تغییرات کل The Total Variation distance

فاصله وَسِراِستین Wasserstein distance

قاعده پس انتشار خطا Backpropagation

کمینه-بیشینه Minimax

کوچک کردن Pooling

گشتاور Momentum

لایه پنهان Hidden Layer

لایه خروجی Output Layer

لایه ورودی Input Layer

لیپشیتز Lipschitz

مسطح سازی Flatten

نظریه بازی Game Theory

واگرایی Divergence

یادگیری عمیق Deep Learning

واژه‌نامه انگلیسی به فارسی

تابع فعالیت(تابع محرک) . . . . . Activation Function

قاعده پس انتشار خطا . . . . . Backpropagation

طبقه‌بندی . . . . . Classification

پیوستگی . . . . . Continuity

شبکه پیچشی . . . . . Convolution Network

شبکه منتقد . . . . . Critic

رویای عمیق . . . . . DeepDream

یادگیری عمیق . . . . . Deep Learning

شبکه متمایزکننده . . . . . Discriminator

واگرایی . . . . . Divergence

شبکه عصبی بازگشتی یا پسخور . . . . . Feed-Back Network

شبکه عصبی ایستا یا پیشخور . . . . . Feed-Forward Network

مسطح سازی . . . . . Flatten

نظریه بازی . . . . . Game Theory

شبکه مولد . . . . . Generative

شبکه مولد رقابتی (مولد متخاصم) . . . . . Generative Adversaial Network(GAN)

لایه پنهان . . . . . Hidden Layer

لایه ورودی . . . . . Input Layer

لیپشیتز . . . . . Lipschitz

تابع هزینه . . . . . Loss Function

کمینه-بیشینه . . . . . Minimax

گشتاور . . . . . Momentum

تعادل نش . . . . . Nash Equilibrium

شبکه عصبی . . . . . Neural Network

تابع بهینه سازی . . . . . Optimizer Function

لایه خروجی . . . . . Output Layer

بیش‌برازش . . . . . Overfit

کوچک کردن . . . . . Pooling

احتمالی . . . . . Probabilistic

پنجره‌ی لغزان . . . . . Sliding Window

تنسور . . . . . Tensor

فاصله تغییرات کل . . . . . The Total Variation distance

داده آموزشی . . . . . Training Data

خودرمزگذار . . . . . Variational Auto-Encoders(VAE)

فاصله وَسِراِستین . . . . . Wasserstein distance

شبکه مولد رقابتی وَسِراِستین . . . . . Wasserstein Generative Adversial Network (WGAN)

بازی مجموع-صفر . . . . . Zero-Sum

Hakim Sabzevari University

An Outline of MSc. Thesis

Surname:Salehi Sadati	Name:Seyedeh Afsaneh	Student No.:9713185015

Supervisors: Dr. Mehdi Zaferanieh and Dr. Mahmood Amintoosi

Advisor: Dr. Amin Rafiei

Faculty of Mathematics and Computer Science

Program: Computer Science Field:Decision Sciences and Knowledge

Title of thesis: Wasserstein Generative Adversial Network

Keywords: Deep Learning, Neural Network, Generative Adversial Network, Game Theory, Discriminator, Nash Equilibrium, Wasserstein Generative Adversial Network

Abstract: Generative Adversial Networks are a kind of learning system for generating new samples of educational data. This method produces new examples of the competition of two neural networks in a game theory. The goal of these networks is to produce new samples with complex distributions such as images. To build these new prototypes, it starts with a simple random noise distribution and uses a neural network to bring it to the desired complex distribution. In this dissertation, we get acquainted with the structure of neural networks and competitive productive networks and examine the examples produced from this network.

Hakim Sabzevari University

Faculty of Mathematics and Computer Science

A Thesis Submitted in Partial Fulfilment of the Requirement for the Degree of Master of Science in Computer Science

Wasserstein Generative Adversial Network

Supervisors:

Dr. Mehdi Zaferanieh and Dr. Mahmood Amintoosi

Advisor:

Dr. Amin Rafiei

By:

Seyedeh Afsaneh Salehi Sadati

August 2020