به نام خدا

درس یادگیری عمیق

تمرین اول

مدرس : دکتر داوودآبادی

دستیاران : حسن حماد، مرتضی حاجی آبادی

دانشجو : سارا سادات یونسی / ۹۸۵۳۳۰۵۳

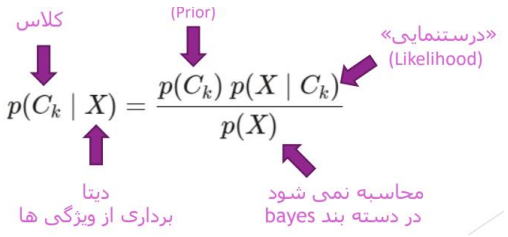
فهرست

۱- الف) در این سوال می خواهیم کلاس متن یک خبر را تشخیص دهیم (کلاس ۰ یا کلاس ۱ .(اطلاعات جدول زیر را در نظر بگیرید و با استفاده از آنها احتمال تعلق خبرهای تست به هر کلاس را محاسبه کنید و کلاس عنوان خبر را با استفاده از مدل های احتمالاتی آموخته شده تشخیص دهید)(۱۰ نمره)

ب) در صورتی که در متن خبر تست، داده جدید داشته باشیم (داده تست دوم) چگونه می توان این داده را تحلیل کرد؟(راهنمایی : استفاده از هموارسازی لاپلاس با ضریب آلفای ۱)(۱۰ نمره امتیازی)

کلاس	متن خبر	نوع داده
۰	فناوری فرهنگی علمی اقتصادی	آموزش
۰	فناوری فرهنگی علمی اجتماعی سیاسی	آموزش
۱	فناوری فرهنگی اجتماعی سیاسی	آموزش
۱	علمی اجتماعی سیاسی اقتصادی	آموزش
؟	فناوری فرهنگی علمی اجتماعی	تست
؟	فناوری فرهنگی علمی اجتماعی ورزشی	تست

قسمت الف)



باتوجه به فرمول بالا و استفاده از دسته بند بیز خواهیم داشت :

$$P(C|D) = P(C)P(D|C)/P(D)$$

C➔کلاس مورد نظر

نمونه تست مورد نظر D➔فناوری ,فرهنگی ,علمی, اجتماعی

مرحله اول :

کلاس ۱ :

$P\ (C = 1|\ D) = P\ (C = 1)\ P(D|C = 1)\ /P(D)$

کلاس ۰ :

$P\ (C = |۰\ D) = P\ (C =0\ )\ P(D|C = ۰)\ /P(D)$

مخرج کسر برای این دوتا کلاس یکسان است ونیازی به محاسبه آن ندارد.

و با توجه به نمونه های آموزش : احتمال انتخاب کلاس ۱= احتمال انتخاب کلاس ۰ = 1/2

$P\ (C =0\ ) = P\ (C =1\ ) =1/2$

مرحله دوم :

کلاس ۰ :

$P(\ D\ |C = 0)$ = P(D1|0) x P(D2|0) x P(D3|0) x P(D4|0)

$P(\ D\ |C = ۰)$ = P(فناوری|۰) x P(فرهنگی|۰) x P(علمی|۰) x P(اجتماعی|۰)

P(فناوری|۰):۰ تعداد آن در کلاس ۰\ تعداد کل ویژگی ها در کلاس ۰ ➔ 2/9

P(فرهنگی|۰):۰ تعداد آن در کلاس ۰\ تعداد کل ویژگی ها در کلاس ۰ ➔ 2/9

P(علمی|۰):۰ تعداد آن در کلاس ۰\ تعداد کل ویژگی ها در کلاس ۰ ➔ 2/9

P(اجتماعی|۰):۰ تعداد آن در کلاس ۰\ تعداد کل ویژگی ها در کلاس ۰ ➔ 1/9

کلاس۱ :

$P(\ D\ |C = 1)$ = P(D1|1) x P(D2|1) x P(D3|1) x P(D4|1)

$P(\ D\ |C = ۱)$ = P(فناوری|۱) x P(فرهنگی|۱) x P(علمی|۱) x P(اجتماعی|۱)

P(فناوری|۱):۱ تعداد آن در کلاس ۱\ تعداد کل ویژگی ها در کلاس ۱ ➔ 1/8

P(فرهنگی|۱):۱ تعداد آن در کلاس ۱\ تعداد کل ویژگی ها در کلاس ۱ ➔ 1/8

P(علمی|۱): تعداد کل ویژگی ها در کلاس ۱\ تعداد آن در کلاس ۱ ➔ 1/8

P(اجتماعی|۱): تعداد کل ویژگی ها در کلاس ۱\ تعداد آن در کلاس ۱ ➔2/8

مرحله سوم :

کلاس ۰ :

$P(C = |۰\ D) = P(C = ۰)\ P(D|C = ۰)$

$P(C = ۰) = 1/2$

$P(C = ۰ | D) = 1/2 * 2/9 * 2/9 * 2/9 * 1/9$ ➔ $8/13122 \sim 0.0006$

کلاس ۱۰ :

$P(C = 1 | D) = P(C = 1)\ P(D|C = 1)$

$P(C = 1) = 1/2$

$P(C = 1| D) = 1/2 * 1/8 * 1/8 * 1/8 * 2/8$ ➔ $2/8192 \sim 0.0002$

احتمال کلاس0 بیشتر از احتمال کلاس 1 پس نمونه تست به کلاس 0 تعلق دارد

برای تست دوم نیز چون

P(ورزشی|۰): 0

P(ورزشی|۱): 0

داداده ی ورزشی در دیتاهای تست وجود ندارد در نتیجه نمی توان کسر مورد نظر را حساب کرد وحاصل ها صفر می شوند.

$P(D|C = ۰)$ = P(فناوری|۰) x P(فرهنگی|۰) x P(علمی|۰) x P(اجتماعی|۰) * P(ورزشی|0):0

$P(D|C = 1)$ = P(فناوری|1) x P(فرهنگی|1) x P(علمی|1) x P(اجتماعی|1) * P(ورزشی|1):0

قسمت ب)

فرمول هموارسازی لاپلاس :

$P(a|X) = X+\alpha/ N+\alpha*k$

در فرمول بالا، آلفا نشان دهنده ضریب، kتعداد feature های دیتا و N نشان دهنده تعداد نتایج با کلاس X هستند. حال با توجه به این فرمول، به محاسبه احتمال های شرطی می پردازیم:

و بعد از بدست آوردن احتمال های شرطی برای هر کدام مراحل را مانند قسمت قبل الف طی می کنیم :

α : 1

k : 6

Nبرای کلاس یک : ۸ Nبرای کلاس صفر : ۹

P(فناوری|۰):2+1/9+6=3/15

P(فرهنگی|۰):2+1/9+6=3/15

P(علمی|۰): 2+1/9+6=3/15

P(اجتماعی|۰): 1+1/9+6=2/15

P(ورزشی|۰): 0+1/9+6=1/15

P(فناوری|۱): 1+1/8+6=2/14

P(فرهنگی|۱): 1+1/8+6=2/14

P(علمی|۱): 1+1/8+6=2/14

P(اجتماعی|۱): 2+1/8+6=3/14

P(ورزشی|۰): 0+1/8+6=1/14

مرحله اول :

کلاس ۱ :

$P\ (C = 1|\ D) = P\ (C = 1)\ P(D|C = 1)\ /P(D)$

کلاس ۰ :

$P\ (C = |۰\ D) = P\ (C =0\ )\ P(D|C = ۰)\ /P(D)$

مخرج کسر برای این دوتا کلاس یکسان است ونیازی به محاسبه آن ندارد.

و با توجه به نمونه های آموزش : احتمال انتخاب کلاس ۱= احتمال انتخاب کلاس ۰ = 1/2

$P\ (C =0\ ) = P\ (C =1\ ) =1/2$

کلاس ۱ :

$P(\ D\ |C = 1)$ = P(D1|1) x P(D2|1) x P(D3|1) x P(D4|1) x P(D5|1)

$P(\ D\ |C = 1)$ = P(فناوری|1) x P(فرهنگی|1) x P(علمی|1) x P(اجتماعی|1) * P(ورزشی|1):

کلاس۰ :

$P(\ D\ |C = 0)$ = P(D1|0) x P(D2|0) x P(D3|0) x P(D4|0) x P(D5|0)

$P(\ D\ |C = ۰)$ = P(فناوری|۰) x P(فرهنگی|۰) x P(علمی|۰) x P(اجتماعی|۰)* P(ورزشی|0):

مرحله سوم :

کلاس ۱ :

$P(\ C = 1|\ D\ )= P(\ C = 1)\ P(D|C = 1)$

$P\ (C = 1) = 1\ /2$

$P(\ C = 1|\ D\ )$=1/2 *2/14*2/14*2/14*3/14 *1/14➔ 2~ 0.00002

کلاس ۰ :

$P(C=0 \mid D) = P(C=0)\, P(D \mid C=0)$

$P(C=0) = 1/2$

$P(C=0 \mid D) = 1/2 * 3/15*3/15*3/15*2/15*1/15 \rightarrow \sim 0.00003$

احتمال کلاس 0 بیشتر از احتمال کلاس 1 پس نمونه تست به کلاس0 تعلق دارد

۲-نوتبوک های آموزشی اجرا را Keras.ipynb و P ytorch.ipynb ، Numpy.ipynb، Basic.ipynbکنید، دقت کنید این نوتبوک ها صرفا برای یادآوری هستند و نمره ای به آن ها تعلق نخواهد گرفت.

۳- n–logistic از تابع سیگموئید

استفاده می کند و سپس تخمینlikelihood maximum را اعمال می کند. می توان از تابع probit (به جای تابع سیگموید) استفاده کرد:

$$\Phi(a) = \int_{-\infty}^{a} N(\theta \mid 0, 1)d\theta$$

که در آن N(teta|0,1) ,توزیع نرمال استاندارد است .برای رگرسیونprobit ، منفی ضررشرطی log − likelihood را محاسبه کنید. نیازی به ساده سازی عبارت نیست.(۱۰ نمره)

$y = xw + b + \epsilon$ ← نویز

bias

$\epsilon \sim G(N(0,1))$

$y \to 1 \to y > 0$

$\to 0 \to y < 0$

از این رابطه برای پیش‌بینی استفاده می‌شود.

$$P(Y=1|x) = Q(x^T\beta) = Q(x_i w + \beta)$$

$$Q(a) = \int_{-\infty}^{a} N(\theta|0,1)\,d\theta$$

$$P(Y=0|x) = 1 - Q(x^T\beta) = 1 - Q(x_i w + \beta)$$

likelihood: $L(\beta; x_i, y_i) = Q(x_i^T\beta)^{y_i} \cdot (1 - Q(x_i^T\beta))^{(1-y_i)}$

OR

$$L(w,b; x_i, y_i) = Q(x_i^T w + b)^{y_i} \times (1 - Q(x_i^T w + b))^{(1-y_i)}$$

x_i: بردار ورودی‌ها β: بردار Coefficient

توزیع نرمال ←

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-1}{2\sigma^2}(y - w^T x - b)^2\right)}$$

احتمال شرطی ←

$$P(y|x) = \prod_{i=1}^{n} P(y_i|x_i)$$

مناسب‌ترین likelihood برای همه‌ی اعضا: مناسب‌ترین انتخاب همه‌ی عناصر مجموعه.

$$L(\beta; X, Y) = \prod_{i=1}^{n} (Q(x_i^T\beta)^{y_i} \cdot (1 - Q(x_i^T\beta))^{(1-y_i)})$$

اعمال روی ln ← برای این‌که log-likelihood به log تبدیل شود.

جواب

$$\sum_{i=1}^{n} (y_i \ln Q(x_i^T\beta) + (1-y_i) \ln(1 - Q(x_i^T\beta)))$$

OR

$$\sum_{i=1}^{n} (y_i \log_e Q(x_i^T w + b) + (1-y_i) \log_e (1 - Q(x_i^T w + b)))$$

۴ -الف) دلیل استفاده از توابع فعال سازی در شبکه های MLP چیست؟

ب) آیا هر تابع غیرخطی را به عنوان تابع فعالسازی میتوان استفاده کرد؟(۱۰ نمره)

قسمت الف)

یک شبکه عصبی بدون تابع فعال سازی در اصل فقط یک مدل رگرسیون خطی است .تابع فعال سازی تبدیل غیرخطی را به ورودی انجام می دهد و آن را قادر به یادگیری و انجام کارهای پیچیده تر می کند.

یک شبکه عصبی بدون تابع فعال‌سازی با محاسبه مجموع وزنی و اضافه کردن بایاس به آن، تصمیم می‌گیرد که آیا نورون باید فعال شود یا نه. هدف از تابع فعال سازی، وارد کردن غیر خطی بودن به خروجی یک نورون است. توضیح: می دانیم که شبکه عصبی دارای نورون هایی است که مطابق با وزن، بایاس و عملکرد فعال سازی مربوطه کار می کنند. در یک شبکه عصبی، وزن‌ها و بایاس‌های نورون‌ها را بر اساس خطای خروجی به‌روزرسانی می‌کنیم. این فرآیند به عنوان پس انتشار شناخته می شود. توابع فعال‌سازی، انتشار به عقب را ممکن می‌سازد، زیرا گرادیان‌ها همراه با خطا برای به‌روزرسانی وزن‌ها و بایاس‌ها ارائه می‌شوند.nction اساساً فقط یک مدل رگرسیون خطی است. تابع فعال سازی تبدیل غیرخطی را به ورودی انجام می دهد و آن را قادر به یادگیری و انجام کارهای پیچیده تر می کند.

به‌طور کلی می‌توان توابع فعالساز (Activation Functions) را به دو دسته تقسیم کرد:

تابع فعال‌ساز خطی(Linear or Identity Activation Function)

توابع فعال‌ساز غیرخطی(Non-linear Activation Functions)

استفاده نکنیم، وزن‌ها و مقدار بایاس فقط یک معادله‌ی خطی را ایجاد می‌کنند. (Activation Functions) اگر از توابع فعالساز درست است که معادله‌ی خطی خیلی راحت‌تر حل‌شدنی است، اما برای حل مسائل پیچیده نمی‌تواند کمکی به ما کند؛ درواقع معادلات خطی در یادگیری الگوهای پیچیده‌ی داده‌ی خیلی محدود هستند و یک شبکه‌ی عصبی بدون تابع فعال‌ساز فقط یک است. به‌طور کلی، شبکه‌های عصبی از توابع فعالساز استفاده می‌کنند تا (Linear Regression Model) مدل رگرسیون خطی .بتوانند به شبکه در یادگیری داده‌های پیچیده کمک و پیش‌بینی قابل‌قبولی را در خروجی ارائه کنند

قسمت ب)

نه هر تابع غیرخطی نمی تواند به عنوان تابع فعال سازی در شبکه های عصبی استفاده شود .

توابع فعال سازی باید دارای ویژگی هایی مانند مشتق قابل محاسبه و تابعیتی باشند که بتوانند به خوبی در فرآیند آموزش شبکه عمل کنند. مانند سیگموید و رلو و ...

باید یکنواخت باسند تا به مینموم محلی برسیم با گرادیان کاهشی باید فعال ساز مشتق پذیر باشد به دلیل back propagation نباید از نز محاسباتی برای ما هزینه زیادو نامعقول نداشته باشد.

- Monotonicity
- کارایی و بهینه بودن محاسبات
- تداوم و تفاوت پذیری
- Risk of overfitŋng
- Non-linearity
- Continuous
- باید ویژگی های بالا را داشته باشد

از ویژگی توابع فعال سازی به موارد زیر می توان اشاره کرد که قابل توجه است که تنها غیر خطی بودن شرط مورد نیاز ما برای انتخاب تابع فعال سازی نخواهد بود

به نظر میرسد که پاسخ به این سوال بستگی به تعریف دقیق تابع غیرخطی و تابع فعالسازی دارد. برخی منابع میگویند که هر تابع غیرخطی که مشتق پذیر باشد، میتواند به عنوان تابع فعالسازی در شبکه های عصبی استفاده شود.

اما برخی منابع دیگر محدودیتهای بیشتری را برای انتخاب تابع فعالسازی مطرح میکنند، مانند اینکه تابع فعال سازی باید پیوسته، چندجمله ای و بازه خروجی محدود داشته باشد.

همچنین برخی خصوصیات دیگر مانند غیر افزاینده بودن، نرمال بودن و داشتن مشتق ساده نیز برای توابع فعال سازی پیشنهاد شده اند.

پس به نظر میآید که نمیتوان گفت هر تابع غیرخطی را میتوان به عنوان تابع فعالسازی در شبکه های عصبی استفاده کرد، بلکه باید از توابع غیرخطی که خصوصیات مناسب را دارند، استفاده کرد.

از جمله ویژگی‌های توابع فعال سازی می‌توان به موارد زیر اشاره کرد:

- غیرخطی بودن: غیرخطی بودن تابع فعال‌سازی به این معنی است که یک شبکه عصبی دو لایه را می‌توان نوعی تابع عمومی تقریب در نظر گرفت. تابع فعال‌سازی identity خطی است. به عبارت دیگر، اگر چندین لایه به صورت همزمان از این تابع فعال‌سازی استفاده کنند، شبکه عصبی به مثابه یک مدل یک لایه عمل خواهد کرد.
- بازه: در صورتی‌که بازه‌ی (Range) تابع فعال‌سازی محدود باشد، متدهای آموزشی مبتنی بر گردایان پایدارتر خواهند بود، چراکه الگوها فقط تعداد کمی از وزن‌ها را تحت تأثیر می‌گذارند. از سوی دیگر، اگر بازه‌ی تابع فعال‌سازی غیرمحدود و نامتناهی باشد، فرایند آموزش کاراتر و اثربخش‌تر خواهد بود، زیرا الگوها بر بیشتر وزن‌ها اثر می‌گذارند. در مورد آخر، به نرخ‌های یادگیری کوچک‌تر نیاز است.
- مشتق‌پذیر پیوسته: در توابع فعال سازی، این ویژگی از آن جهت مطلوب است که در متدهای بهینه‌سازی مبتنی بر گرادیان کاربرد دارد (تابع ReLU مشتق‌پذیر پیوسته نیست و به همین دلیل در بهینه‌سازی مبتنی بر گرادیان به مشکل می‌خورد .(تابع پله‌ای دودیی در ۰ مشتق‌پذیر نیست و برای تمامی مقادیر در ۰ مشتق می‌شود، به همین دلیل آن دسته از متدهای مبتنی بر گرادیان که از این تابع استفاده می‌کنند، پیشرفتی نخواهند داشت.

مشتق، مشتق‌پذیری یا شیب: زمانی‌که در محور y تغییر کنند، در محور x هم تغییر می‌کنند.

- یکنوا :در صورتی‌که تابع فعال‌سازی یکنوا باشد، متغیر خطای مدل یک لایه محدب خواهد بود.

تابع یکنوا: به تابعی گفته می‌شود که یا صعودی است یا نزولی.

- توابع هموار با مشتق یکنوا: این نوع توابع در برخی موارد عملکرد بهتری در امر تعمیم‌دهی دارند.
- موجودیت را نزدیک به ریشه یک تابع (صفرها) تقریب می‌زند: چنانچه تابع این ویژگی را داشته باشد و وزن‌های شبکه عصبی با مقادیر تصادفی مقداردهی شوند، یادگیری شبکه کارا خواهد بود. اگر تابع فعال‌سازی موجودیت را نزدیک به ریشه تقریب نزند، باید در مقداردهی وزن‌های توجه بیشتری به خرج دهیم.

---

۵ -الف)توابع فعال سازی زیر را توضیح دهید (با ذکر مزایا و معایب هر کدام) و باهم دیگر مقایسه کنید .

•تابع سیگموئید:

•تابع :softmax

•تابع Relu

•تابع :Tanh

ب)در این بخش از سوال می خواهیم توابع فعال سازی را پیاده سازی کنیم و با توابع فعال سازی کتابخانه ytorch P مقایسه کنیم. برای این کار نوتبوک ipynb.functions_activation را تکمیل کنید.

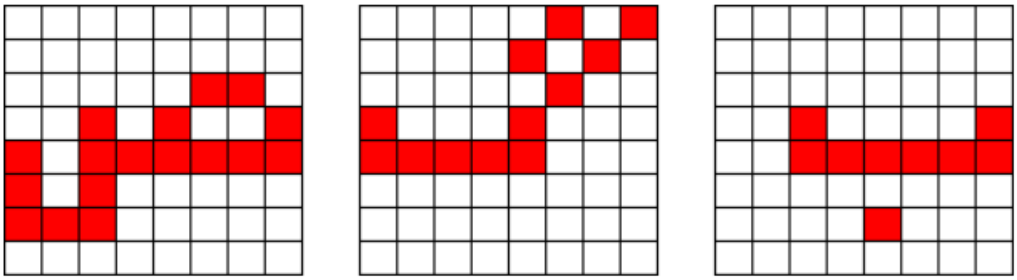
ج)حال می خواهیم یک شبکه MLP ساده طراحی کنیم که بتواند تصاویر زیر را از هم دیگر جدا کند. برای این کار یک معماری برای شبکه MLP ارائه دهید و علت انتخاب این معماری را توضیح دهید .معماری شما باید شامل اجزای زیر می شود :

•تعداد لایه ها و علت انتخاب این تعداد

•تعداد نورون های هر لایه و علت انتخاب این تعداد

•تابع فعال سازی و علت انتخاب آن

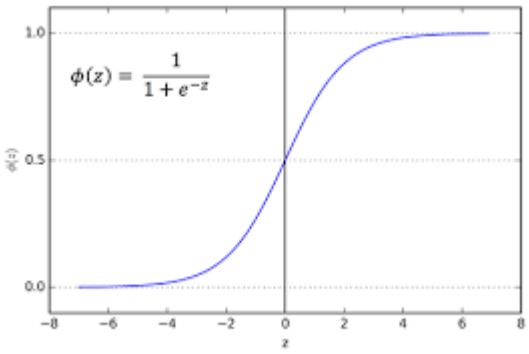
•تابع ضرر و علت انتخاب آن



د)برای اطمینان بیشتر، یک شبکه MLP طبق معماری خود را با استفاده از کتابخانه Pytorch پیاده سازی کنید. سپس با استفاده از تصاویر مورد نظر تست کنید و نتایج را گزارش کنید.(۳۰ نمره)

قسمت الف)

- **تابع سیگموئید:**



ین تابع یک منحنی S شکل است. زمانی‌که می‌خواهیم خروجی مدل احتمال باشد، از تابع سیگموید استفاده می‌کنیم؛ چون تابع سیگموید مقادیر را به بازه صفر تا ۱ می‌برد و احتمالات هم میان همین بازه قرار دارند.

مزایا

این تابع تمایزپذیر (Differentiable) است؛ یعنی در هر قسمت از منحنی می‌توانیم شیب میان دو نقطه را حساب کنیم.

از آنجا که این تابع مقادیر را میان صفر و یک قرار می‌دهد، نوعی عادی‌سازی را برای خروجی هر نورون انجام می‌دهد.

معایب

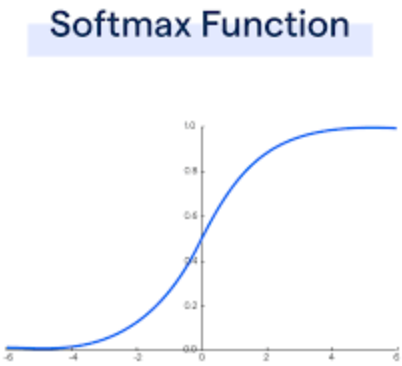
با محوشدگی گرادیان (Vanishing Gradient) مقادیر بسیار بزرگ یا بسیار کوچکx ، مشتق بسیار کوچک می‌شود و درواقع شبکه دیگر آموزش نمی‌بیند و پیش‌بینی‌هایش در خروجی ثابت می‌ماند.

به‌دلیل مشکل محوشدگی گرادیان، تابع سیگموید هم‌گرایی کند دارد.

خروجی تابع سیگموید صفرمحور (Zero-Centered) نیست؛ این امر کارایی به‌روزرسانی وزن‌ها را کم می‌کند.

از آنجا که این تابع عملیات نمایی (Exponential Operations) دارد، می‌توان گفت هزینه‌ی محاسباتی بالایی دارد و کندتر پیش می‌رود.

- تابع softmax:



تابع فعالساز از جمله توابع فعالساز (Activation Functions) است که در طبقه‌بندی‌های چندکلاسه استفاده می‌شود. زمانی‌که احتیاج داشته باشیم در خروجی احتمال عضویت بیشتر دو کلاس را پیش‌بینی کنیم، می‌توانیم به‌سراغ این تابع برویم. تابع سافت‌مکس تمامی مقادیر یک بردار با طول K را به بازه‌ی صفر تا ۱ می‌برد، به‌طوری که جمع تمامی مقادیر این بردار با هم ۱ می‌شود. این تابع برای نورون‌های لایه‌ی خروجی استفاده می‌شود؛ زیرا در شبکه‌های عصبی در آخرین لایه (خروجی) به طبقه‌بندی ورودی‌ها در کلاس‌های مختلف نیاز داریم.

مزایا

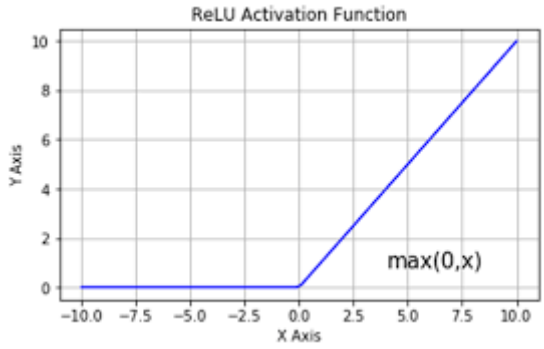
این تابع قابلیت استفاده در تسک های چندکلاسه را دارد. خروجی هر کلاس را میان صفر تا ۱ عادی‌سازی می‌کند؛ سپس آن‌ها را بر مجموعه‌شان تقسیم و احتمال عضویت مقادیر ورودی را در هر کلاس به ما در خروجی ارائه می‌کند.

معایب

مقدار گرادیان برای مقادیر منفی صفر است؛ به‌این معنا که وزن‌ها در حین عملیات پس‌انتشار به‌روزرسانی نمی‌شوند و این می‌تواند مشکل مرگ نورون را ایجاد کند.

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for j} = 1,\ldots,\text{K}$$

•تابع Relu



تابع فعالساز واحد یک‌سوشده‌ی خطی در زمینه‌ی یادگیری عمیق بسیار مشهور است و در بیشتر مواقع استفاده می‌شود. این تابع به‌این صورت عمل می‌کند که مقادیر منفی (زیر صفر) را صفر و مقادیر مثبت (بیشتر از صفر) و مقادیر برابر با صفر را همان مقدار خودش در نظر می‌گیرد.

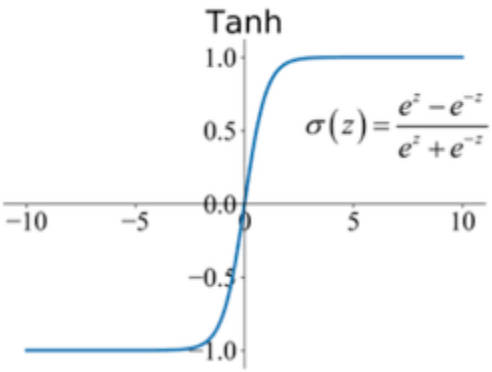
مزایا:

- از نظر محاسباتی بسیار کارآمد است و به شبکه اجازه می‌دهد به‌سرعت همگرا شود؛ زیرا رابطه‌ی آن خطی است و به‌همین دلیل، در مقایسه با تابع‌های سیگموید وTanh ، سریع‌تر است.

معایب:

- مشکل مرگ نورون یا مرگ ReLU دارد؛ یعنی زمانی‌که ورودی صفر یا نزدیک به صفر باشد، تابع ReLU دیگر عملکردی ندارد و به‌بیان دیگر، می‌میرد. در این صورت، مقدار گرادیان تابع صفر می‌شود و شبکه نمی‌تواند عملیات پس انتشار (Backpropagation) را انجام دهد و آموزش ببیند.
- خروجی این تابع صفر یا مثبت است و این یعنی صفرمحور نیست.

- تابع Tanh:



این تابع هم مانند تابع سیگموید به‌شکل S است، اما در مقایسه با تابع سیگموید، نکات مثبت بیشتری دارد.

مزایا

این تابع صفرمحور است؛ بنابراین به مدل کمک می‌کند تا مقادیر ورودی منفی، خنثی و مثبت داشته باشد؛ به‌عبارت دیگر، مقادیر منفی، به‌شدت منفی و مقادیر صفر در گراف تانژانت هایپربولیک نزدیک به صفر نگاشت می‌شوند.

تابع آن یکنواخت(Monotonic) ، اما مشتق آن یکنواخت نیست.

معایب

محوشدگی گرادیان

هم‌گرایی کند

فرق بین تابع sigmoid وsoftmax

۱. هر دو تابع softmax و sigmoid توابع ریاضی هستند که در یادگیری ماشین و شبکه‌های عصبی مصنوعی برای اهداف مختلف استفاده می‌شوند.
۲. تابع sigmoid یک تابع فعالساز رایج است که هر عدد را می‌گیرد و آن را به مقداری بین ۰ و ۱ ترسیم می‌کند.
۳. Sigmoidاغلب در مسائل طبقه‌بندی باینری استفاده می‌شود، جایی که خروجی مدل باید به عنوان احتمال کلاس مثبت تفسیر شود. با این حال، سیگموید از مشکل محوشدگی گرادیان رنج می‌برد، که می‌تواند آموزش شبکه‌های عصبی عمیق‌تر را دشوار کند.

از طرف دیگر، تابع softmax یک تابع کلی‌تر است که می‌تواند برای تبدیل بردار اعداد واقعی به توزیع احتمال استفاده شود. یک بردار ورودی می‌گیرد و یک توزیع احتمال روی K کلاس را در خروجی می‌دهد، که در آن K تعداد کلاس‌ها است.

Softmaxاغلب به عنوان تابع فعال‌ساز لایه خروجی در مسائل طبقه‌بندی چند کلاسه استفاده می‌شود، جایی که هدف، پیش‌بینی احتمال هر کلاس است. همچنین در پردازش زبان طبیعی برای مدل‌سازی زبان و وظایف طبقه‌بندی متن استفاده می‌شود.

به طور خلاصه، sigmoidیک تابع فعالساز باینری است که در مسائل طبقه‌بندی باینری استفاده می‌شود، در حالی که softmaxیک تابع فعالساز چند کلاسه است که در مسائل طبقه‌بندی چند کلاسه استفاده می‌شود.

مقایسه RElu و tanh :

توابع فعال سازی tanh و ReLU دو انتخاب رایج برای شبکه های عصبی هستند .آنها خواص و مزایای مختلفی دارند که آنها را برای کارهای مختلف مناسب می کند .در اینجا مقایسه مختصری از این دو عملکرد وجود دارد :

•تابع tanh یک تابع مماس هذلولی است که مقداری بین ۱- و ۱ را خروجی می دهد. این تابع یکنواخت و غیرخطی است که می تواند هر تابع پیوسته را تقریبی کند .در اطراف مبدا متقارن است و میانگین صفر دارد که می تواند به فرآیند بهینه سازی کمک کند

•تابع ReLU یک تابع واحد خطی اصلاح شده است که حداکثر مقدار بین صفر و ورودی را خروجی می کند .این یک تابع خطی و غیرخطی تکه ای است که می تواند نمایش های پراکنده را یاد بگیرد .نامتقارن است و میانگین مثبتی دارد، که می تواند به پراکندگی و استحکام شبکه کمک کند که در شبکه عصبی ساده، بهتر از relu-relu عمل کند .برخی از مزایای استفاده از tanh نسبت به ReLU عبارتند از :

Tanh •می تواند ورودی های منفی را بهتر از ReLU کنترل کند، که می تواند باعث غیرفعال شدن برخی نورون ها و تولید خروجی صفر شود .این می‌تواند به مشکل نورون مرده منجر شود، جایی که برخی از نورون‌ها به هر ورودی پاسخ نمی‌دهند و برای یادگیری عملکرد بهتر از relu در شبکه عصبی ساده بی‌فایده می‌شوند .

Tanh •می تواند گرادیان های صاف تری نسبت به ReLU ایجاد کند که می تواند در صفر ناپیوستگی داشته باشد .این می تواند به پایداری و همگرایی شبکه کمک کند برخی از مزایای استفاده از ReLU نسبت به tanh عبارتند از :

ReLU •بهتر از tanh می تواند از مشکل گرادیان ناپدید شدن جلوگیری کند، که می تواند گرادیان های بسیار کوچکی برای ورودی های بزرگ یا کوچک داشته باشد .این باعث می شود شبکه نتواند به طور موثر یاد بگیرد

ReLU •می تواند سریعتر از tanh محاسبه شود، که می تواند شامل عملیات ریاضی پیچیده تری باشد .این می تواند در زمان و منابع شبکه صرفه جویی کند بسته به مشکل و داده ها، tanhیا ReLU ممکن است بهتر از دیگری عمل کنند .هیچ پاسخ قطعی برای اینکه کدام یک برتر است وجود ندارد، زیرا هر دو نقاط قوت و ضعف خود را دارند .برخی از عواملی که ممکن است بر انتخاب عملکرد فعال سازی تأثیر بگذارند عبارتند از • :محدوده و توزیع مقادیر ورودی •تعداد و اندازه لایه های پنهان •روش های مقدار اولیه یادگیری و وزن

انواع تفاوت ها :

۱. تابع سیگموید (Sigmoid) در مسائل طبقه‌بندی معمولاً خیلی خوب عمل می‌کند.
۲. توابع سیگموید (Sigmoid) و تانژانت هایپربولیک(Tanh) ، به‌دلیل مشکل محوشدگی گرادیان، در بعضی مواقع استفاده نمی‌شوند.
۳. تابع فعالساز واحد یک‌سوشده‌ی خطی (ReLU) بیشتر از باقی استفاده می‌شود و نتایج خوبی را در خروجی ارائه می‌کند.
۴. تابع فعالساز واحد یک‌سوشده‌ی خطی (ReLU) فقط در لایه‌های نهان (Hidden Layers) استفاده می‌شود.
۵. اگر با مشکل مرگ نورون در شبکه مواجه هستیم، تابع Leaky ReLU می‌تواند گزینه‌ی بسیار خوبی باشد.
۶. تابع تانژانت هایپربولیک(Tanh) ، به‌دلیل مشکل مرگ نورون، کمتر استفاده می‌شود.

Function	Range	0-centered	Saturation	Vanishing Gradient	Computation
Sigmoid	0,1	No	For negative and positive values	Yes	Compute-intensive
Tanh	-1,1	Yes	For negative and positive values	Yes	Compute-intensive
ReLu	0, +∞	No	For negative values	Yes (Better than sigmoid and tanh)	Easy to compute
Leaky ReLu	-∞, +∞	Close	No	No	Easy to compute

Activation Function	Formula	Common Applications	Advantages	Disadvantages
Linear	$f(x) = x$	Rarely used in modern neural networks	None	Does not introduce non-linearity
Binary	$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$	Classification tasks with binary output	Simple and efficient	Limited to binary classification tasks
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Classification tasks	Output can be interpreted as probability	Saturates for large positive or negative inputs; not zero-centered
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Classification tasks	Zero-centered; does not saturate for large positive or negative inputs	None
ReLU	$f(x) = \max(0, x)$	Widely used in modern neural networks	Simple and efficient; does not saturate for large inputs	Can suffer from "dying ReLU" problem; not zero-centered
Softmax	$f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$	Classification tasks with multiple classes	Outputs probability distribution over classes	None

قسمت ب)

کد کامل شده در فایل ها موجود است.

اجرا در محیط colab

در این کد، شما یک تابع به نام sigmoid تعریف میکنید که یک پارامتر به نام x میگیرد. این تابع یکی از معروفترین توابع فعالسازی در شبکههای عصبی - این تابع یک ورودی حقیقی را گرفته و خروجی آن را به بازه [۰, ۱] فشرده میکند. این خروجی میتواند به عنوان احتمال یا درجه اطمینان تفسیر شود

در کد شما، شما از تابع torch.exp استفاده میکنید که عدد طبیعی e را به توان x میرساند. سپس شما از عملگرهای جبر خطی استفاده میکنید تا برابر عبارت بالا را بدست آورید. خروجی این تابع یک تنسور پایتورچ با همان شکل ورودی است.

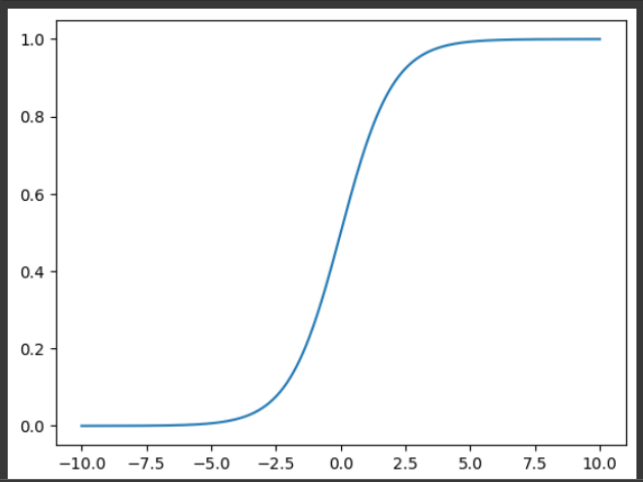
Sigmoid

```
[8]  # implement the sigmoid activation function
     def sigmoid(x):
       return 1/(1 + torch.exp(-x))
```

```
[10] result_sigmoid, output_sigmoid = test(test_data_torch, sigmoid, nn.Sigmoid())
     print(result_sigmoid)
```

```
True
```

```
# plot the sigmoid activation function

plot_function(test_data_np, output_sigmoid)
```

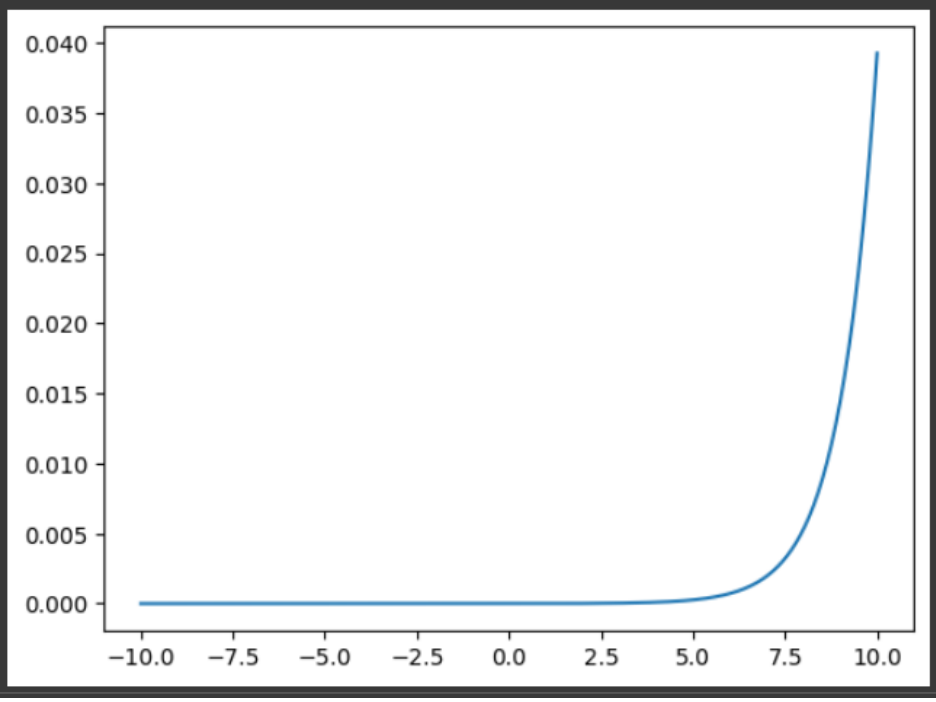


```
[12] # implement the softmax activation function
     def softmax(x):
         return torch.exp(x) / torch.sum(torch.exp(x))
```

```
[13] # test your function against the softmax activation function from torch
     result_softmax, output_softmax = test(test_data_torch, softmax, nn.Softmax())
     print(result_softmax)
```

```
True
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:1518: UserWarning: Impl
  return self._call_impl(*args, **kwargs)
```

```
# plot the softmax activation function
plot_function(test_data_np, output_softmax)
```

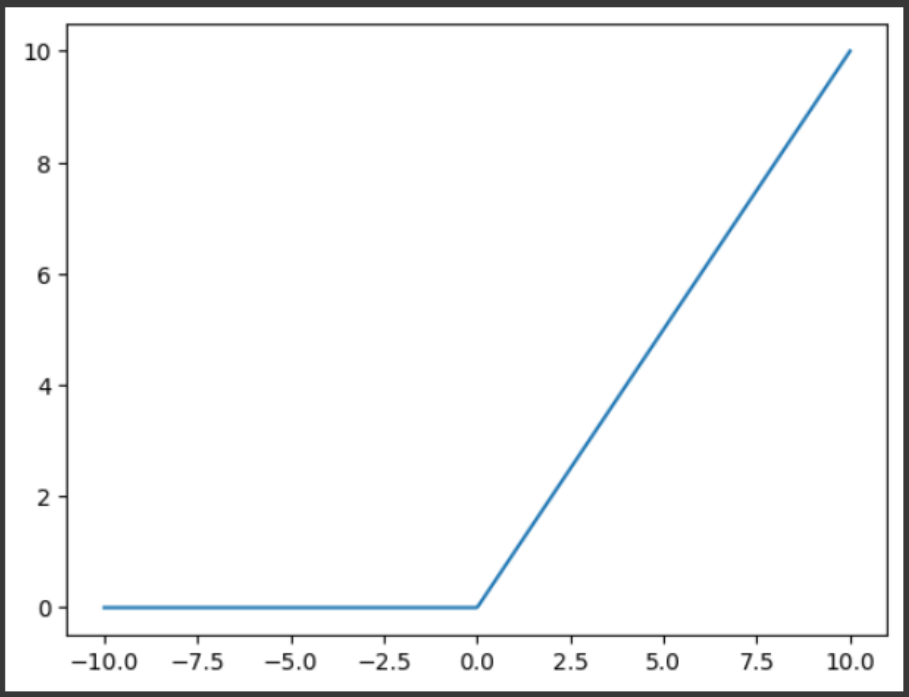


در این کد، شما یک تابع به نام softmax تعریف میکنید که یک پارامتر به نام x میگیرد. این تابع یکی از معروفترین توابع فعالسازی در شبکههای عصبی است. این تابع برای مسائل دسته بندی چند کلاسه مناسب است زیرا خروجی آن را به عنوان توزیع احتمال قابل تفسیر میکند. فرمول این تابع به صورت زیر است:

این تابع برای هر کلاس یک احتمال را حساب میکند که جمع آنها برابر با ۱ است. در کد شما، شما از تابع torch.exp استفاده میکنید که عدد طبیعی e را به توان x function. سپس شما از تابع torch.sum استفاده میکنید که جمع عناصر یک تنسور را در یک بعد خاص محاسبه میکند. در نهایت شما از عملگر تقسیم

(/) استفاده میکنید تا برابر عبارت بالا را بدست آورید. خروجی این تابع یک تنسور پایتورچ با همان شکل ورودی است.

```
[19] # implement the ReLU activation function
     def Relu(x):
       return torch.max(torch.zeros(x.shape), x)

[20] # test your function against the ReLU activation function from torch
     result_relu, output_relu = test(test_data_torch, Relu, nn.ReLU())
     print(result_relu)

     True

     # plot the ReLU activation function
     plot_function(test_data_np, output_relu)
```

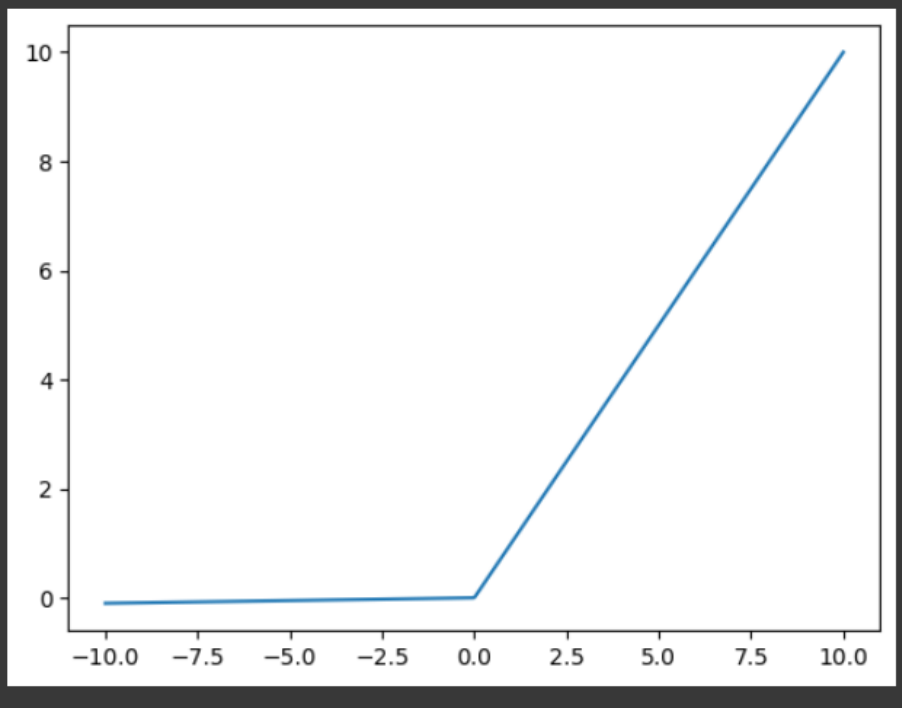


در این کد، شما یک تابع به نام Relu تعریف میکنید که یک پارامتر به نام x میگیرد. این تابع یکی از معروفترین توابع فعالسازی در شبکههای عصبی ا. این تابع برای هر ورودی، حداکثر آن را با صفر برمیگرداند. فرمول این تابع به صورت زیر است:

این تابع خاصیت غیرخطی را به مدل اضافه میکند و باعث میشود که بخشی از نورونها غیرفعال شوند و در نتیجه تنوع و مقاومت در لایههای پنهان افزایش یابد. در کد شما، شما از تابع torch.max استفاده میکنید که حداکثر دو تنسور را برمیگرداند. شما همچنین از تابع torch.zeros استفاده میکنید که یک تنسور صفر با شکل داده شده را برمیگرداند

. در نهایت شما با استفاده از عملگر (,) دو تنسور x و torch.zeros(x.shape) را به تابع torch.max میدهید و خروجی آن را برمیگردانید. خروجی این تابع یک تنسور پایتورچ با همان شکل ورودی است

```
[22] # Implement the Leaky ReLU activation function
     def LeakyRelU(x):
         s=0.01 * x
         return torch.max(s, x)

[23] # test your function against the Leaky ReLU activation function from torch
     result_leaky_relu, output_leaky_relu = test(test_data_torch, LeakyRelU, nn.LeakyReLU())
     print(result_leaky_relu)

     True

     # plot the Leaky ReLU activation function
     plot_function(test_data_np, output_leaky_relu)
```



ما تابع فعال سازی ReLU یک مشکل دارد: ممکن است باعث شود برخی از نورون ها غیرفعال شوند و وقتی ورودی های آن ها منفی باشند، دیگر چیزی یاد نگیرند. این به این دلیل است که گرادیان تابع ReLU برای ورودی های منفی صفر است، که به این معنی است که هیچ سیگنال خطایی نمی تواند از طریق آن نورون ها پس انتشار یابد.

برای حل این مشکل، تابع فعال سازی LeakyReLU یک شیب کوچک برای ورودی های منفی معرفی می کند، به جای صفر صاف. ضریب شیب قبل از آموزش تعیین می شود، یعنی در طول آموزش یادگیری نمی شود. تابع فعال ساز LeakyReLU به صورت زیر تعریف می شود

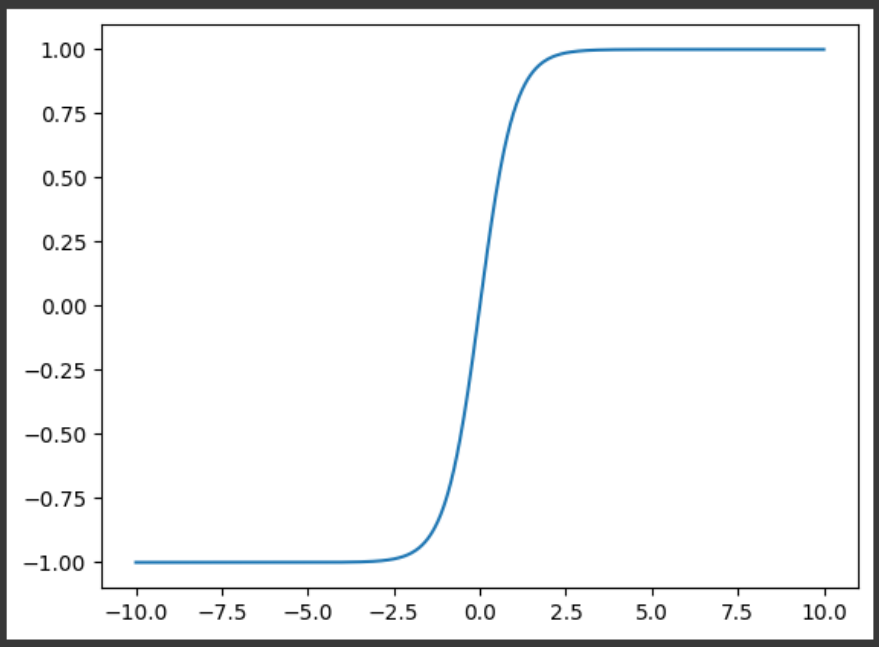
Tanh

```
[28] # implement the tanh activation function
     def Tanh(x):
         s=-2*x
         v=torch.exp(s)
         q=((1-v)/(1+v))
         return (q)
```

```
[29] # test your function against the tanh activation function from torch
     result_tanh, output_tanh = test(test_data_torch, Tanh, nn.Tanh())
     print(result_tanh)
```

```
True
```

```
# plot the tanh activation function
plot_function(test_data_np, output_tanh)
```



تابع Tanh(x) که شما نوشته اید، یک پیاده سازی از این فرمول در زبان برنامه نویسی پایتورچ است. پایتورچ یک کتابخانه علمی برای یادگیری عمیق و محاسبات تانسور است. شما از تابع torch.exp(s) برای محاسبه قدرت نمایی e^s استفاده کرده اید. شما همچنین از عملگرهای جبر خطی برای انجام عملیات بر روی تانسورها استفاده کرده اید.

بنابراین، تابع Tanh(x) شما به این صورت عمل می کند:

- ابتدا x را در -۲ ضرب می کند و نتیجه را در s ذخیره می کند.
- سپس قدرت نمایی e^s را با استفاده از تابع torch.exp(s) محاسبه می کند و نتیجه را در v ذخیره می کند.
- سپس با استفاده از عملگر جبر خطی (/)، v-۱ را بر v+۱ تقسیم می کند و نتیجه را در q ذخیره می کند.
- در نهایت q را به عنوان خروجی تابع بر می گرداند

قسمت ج)

•تعداد لایه ها و علت انتخاب این تعداد :

طور کلی، هر چه تعداد لایهها و عصبها در هر لایه بیشتر باشند، شبکه عصبی پیچیدهتر و قویتر میشود، اما همچنین خطر بیشبرازش (overfitting) را نیز افزایش میدهد وقتی دیتای کمی داریم. بنابراین، باید یک تعادل مناسب بین سادگی و پیچیدگی شبکه عصبی پیدا کنید.۳ لایه برای حل این مسئله کافی می باشد.. همچنین حداقل تعداد لایه هارا باید در نظر بگیریم تا باعث کندی شبکه و هزینه های اضافی نشویم.

لایه ی ورودی : وارد کردن اینپوت های مورد نظر به برنامه تصاویر مورد نظر و خروجی و کلاس بندی مورد انتظار

لایه ی پنهان : برای اعمال توابع فعال سازی و غیرخطی کردن

لایه ی خروجی : که شامل کلاس بندی کردن داده های ما و سه کلاس اصلی حروف

•تعداد نورون های هر لایه و علت انتخاب این تعداد

تعداد نورون لایه ی اول : 64 تعداد پیکسل های ورودی

تعداد نورون لایه ی پنهان : ۳۲ یا ۱۶ چون در نهایت ما یک طبقه بندی سه کلاسه داریم که با تعداد دیتاست کم این کار صورت می گیرد و می تواند تعداد ان از نورون های ورودی کم تر باشد با تجربه و try های مختلف می توان به هیدن لیرر ۱۶ رسید.

تعداد نورون های لایه ی آخر : ۳ عدد می باشد که چون سه حرف داریم و حرف شناسایی شده متناظر با یکی از این سه کلاس می باشد.

•تابع فعال سازی و علت انتخاب آن

استفاده از Relu و Softmax

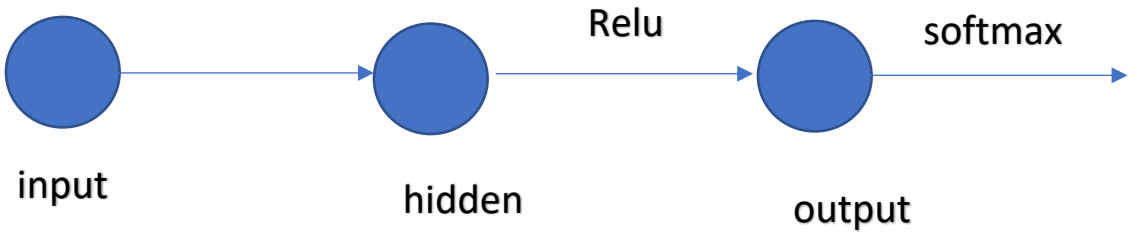
Relu یکی از توابع فعال سازی است که در شبکه های عصبی برای اضافه کردن خاصیت غیرخطی به مدل استفاده می شود.
Relu تابعی ساده است که برای ورودی های مثبت خود را برابر با ورودی قرار می دهد و برای ورودی های منفی صفر را برمی گرداند.

بسیار سریع می باشد دارای پیچدگی محاسباتی پایینی است که در نتیجه باعث عملکرد خوب ان در شبکه می شود و بهنگی را با سرعت بسیار بالا دنبال می کند.

پس انتخاب مناسبی برای شبکه ما می باشد همچنین ناپدید شدن گرادیان ها نیز کمتر در ان رخ می دهد .

می توان برای لایه ی خروجی ما که انتهای ان سه کلای مورد نظر ما هست نیز softmaxانتخاب کنیم و خروجی را نرمال کنیم.
که برای طبقه بندی مسائل چند کلاسه هم استفاده می شود استفاده کنیم . اما هزینه محاسباتی مار ار افزایش می دهد و از ان یک توزیع احتمالاتی بین سه کلاس و انتخاب کلاس با حتمال بالا تر را نشان می دهد و یا با محاسبه ی ماکس logit به جواب نهایی خود خواهیم رسید.

نبود سافت مکس باعث افزایش واریانس میشود و نرمال سازی انجام نمی شود.



•تابع ضرر و علت انتخاب آن :

برای اینکه این مسئله یک مسئله دسته بندی چند کلاسه می باشد می توانیم از تابع CrossEntropy استفاده کنیم.

در کلاسیفیکیشن (single-label classification) خروجی مورد انتظار یک بردار one-hot می باشد که فقط کافیست یک مقداری از خروجی (مقدار ماکزیمم) که متناظر آن در برچسب یک است را در نظر بگیریم و مقدار بقیه خروجی ها اهمیتی ندارد. برای این منظور بهتر است از Cross-E استفاده کنیم.

در کلاسیفیکیشن (multi-label classification) از logistic regression loss استفاده می شود. که همون cross-E هستش که هم مقادیر صفر و هم مقادیر یک را در نظر میگیرد.

بع تلفات متقاطع آنتروپی روشی متداول برای اندازه گیری عملکرد یک مدل طبقه بندی است که یک مقدار احتمال بین ۰ و ۱ را تابع از دست .همچنین به عنوان از دست دادن گزارش یا احتمال ورود منفی شناخته می شود .برای هر کلاس خروجی می دهد دادن آنتروپی متقاطع، میانگین تعداد بیت های مورد نیاز برای شناسایی یک رویداد را از مجموعه ای از رویدادهای ممکن، بر هرچه تلفات متقاطع آنتروپی کمتر باشد، مدل در پیش‌بینی .اساس توزیع احتمال پیش بینی شده توسط مدل، محاسبه می کند تابع از دست دادن آنتروپی متقاطع را می توان برای مسائل طبقه بندی باینری، جایی که فقط دو کلاس .کلاس صحیح بهتر است ممکن وجود دارد، یا برای مسائل طبقه بندی چند طبقه، که بیش از دو کلاس وجود دارد، اعمال کرد

.همچنین برای بهینه سازی از روش های گوناگونی می توان استفاده کرد که یکی از روش های مناسب آدام می باشد

قسمت د)

همانگونه که پیش بینی می کنیم در مرحله ی train کاهش lossرا مشاهده می کنیم تا جایی که در ایپوک ها به صفر برسد و می دانیم که MLP ما به راحتی می تواند این مسئله را حل کند و در مرحله ی تست هم همه را درست پیش بینی کند.

مدل را با استفاده از چند ایپوک آموزش می دهیم .

برای ترین مدل از ۱۰۰ ایپوک استفاده می کنیم و می بینیم لاس کاهش یافته و بعد مدل کار کرده است از ۱,۹۲ به ۰,۸۴۹ رسیده است.

ابتدا کتابخانه های پایتورچ را ایمپورت کردیم وبعد عکس ها را از فایل می خوانیم و تبدیل به grayscale و تبدیل به تصویر ۸*۸ بعد در کلاس main_l لایه ها و تعداد نورون ها و در نهایت مسیر پیش روی شبکه را مشخص کردیم.

که از چه توابع فعال سازی باید در نهایت عبور کند .

و بعد تابع ضرر و بهینه سازی با لرنینگ ریت مشخص را پیاده سازی کردیم و ابرپارامتر خود را ست کردیم و بعد برای آموزش شبکه دیتاهای ترین را reshapeکردیم و قابل استفاده تنسور کردیم.

سلام، این بینگ است. من میتوانم به شما در مورد کد پایتورچ کمک کنم. blush

سه تصویر را با استفاده از یک تبدیل (transform) که قبلاً تعریف کردهاید، به تنسورهای پایتورچ تبدیل میکنید. تبدیل شما شامل سه مرحله است:

- transforms.Resize((۸, ۸)): این مرحله اندازه هر تصویر را به ۸ در ۸ پیکسل تغییر میدهد. این کار برای کاهش ابعاد دادهها و سرعت بخشیدن به آموزش شبکه عصبی مفید است

- transforms.Grayscale(num_output_channels=1): این مرحله هر تصویر را به حالت خاکستری (grayscale) تبدیل میکند. این کار برای حذف اطلاعات رنگی از تصاویر و تمرکز بر ویژگیهای دیگر آنها مفید. پارامتر num_output_channels=1 نشان میدهد که خروجی این مرحله یک کانال دارد.

- transforms.ToTensor(): این مرحله هر تصویر را از یک شئ PIL Image یا numpy array به یک تنسور پایتورچ تبدیل میکند. این کار برای استفاده از قابلیتهای پایتورچ در آموزش شبکه عصبی ضروری است. این مرحله همچنین دادههای تصویر را از بازه [۰, ۲۵۵] به بازه [۰,۰, ۱,۰] نرمال میکند.

پس از این که تبدیل را ساختید، دارید آن را بر روی سه تصویر image1، image2 و image3 اعمال میکنید و نتایج را در سه تنسور tensor_image1، tensor_image2 و tensor_image3 ذخیره میکنید. هر یک از این تنسورها شکل (۱, ۸, ۸) دارند، که نشان میدهد که هر تصویر ۱ کانال و ۸ در ۸ پیکسل است.

سپس دارید هر یک از این تنسورها را با استفاده از دستور print در خروجی نشان میدهید. خروجی هر تنسور شامل ۸ سطر و ۸ ستون است که نشان دهنده مقادیر پیکسلهای هر تصویر است. هر پیکسل یک عدد حقیقی بین صفر و یک است که نشان دهنده شدت روشنایی آن است.

در نهایت، شما یک تنسور دیگر به نام y میسازید که حاوی برچسبهای هر تصویر است. شما برچسبهای ۰، ۱ و ۲ را به عنوان خروجی مورد

در این کد، دارید سه تصویر را با استفاده از تابع torch.stack به یک تنسور چند بعدی تبدیل میکنید. این تابع یک لیست از تنسورها را گرفته و آنها را در یک بعد جدید به هم متصل میکند. برای این کار، همه تنسورها باید از اندازه یکسان باشند

train_data را ساختید، دارید آن را با استفاده از تابع torch.Tensor.view به شکل دیگری تغییر میدهید. این تابع یک تنسور را گرفته و آن را با حفظ دادههای داخل آن، به شکل جدیدی برمیگرداند. در اینجا، شما train_data را به شکل (train_data.size(0), -1) میبرید. این به این معنی است که شما میخواهید تعداد سطرهای train_data را حفظ کنید (که برابر با تعداد تصاویر است) ولی ستونهای آن را به گونهای تغییر دهید که همه پیکسلهای هر تصویر در یک سطر قرار بگیرند. علامت -۱ به پایتورچ میگوید که خودش ستونهای لازم را محاسبه می کنیم.

در نهایت، شما یک تنسور دیگر به نام train_labels میسازید که حاوی برچسبهای هر تصویر است. شما از نوع LongTensor استفاده میکنید که یک نوع خاص از تنسور است که دادههای عدد صحیح طولانی (۶۴ بیت) را نگه میدارد. شما برچسبهای ۰، ۱ و ۲ را به عنوان خروجی مورد نظر خود برای هر تصویر وارد میکنید.

تمام دادههای آموزشی است و بهینهسازی وزنهای مدل را با استفاده از الگوریتم بهینهساز و تابع هزینه انجام میدهد. در اینجا، مراحل کد را توضیح میدهم:

- for epoch in range(num_epochs): این خط یک حلقه برای تعداد دورههای آموزشی که قبلاً تعریف شدهاند، ایجاد میکند. در هر دوره، مقدار خطا (loss) و بهینه ساز (optimizer) بروزرسانی میشوند.

- loss = loss_function(model(train_data), train_labels): این خط خطا را با استفاده از تابع هزینه که قبلاً تعریف شده است، محاسبه میکند. تابع هزینه یک عدد حقیقی است که نشان میدهد که چقدر خروجی مدل با برچسبهای واقعی فاصله دارد. هر چه خطا کمتر باشد، یعنی مدل بهتر عمل کرده است.

- optimizer.zero_grad(): این خط گرادیانهای قبلی را صفر میکند. گرادیان نشان میدهد که چگونه وزنهای مدل را تغییر دهیم تا خطا را کمتر کنیم. در پایتورچ، گرادیانها به صورت خودکار در هر بار فراخوانی تابع backward() جمع شده و بروزرسانی میشوند. بنابراین، قبل از هر دوره آموزشی، باید گرادیانهای قبلی را صفر کنید تا تاثیر آنها روی گرادیان جدید وجود نداشته باشد .

- loss.backward(): این خط گرادیان جدید را با استفاده از قانون زنجیرهای (chain rule) محاسبه و ذخیره میکند

- optimizer.step(): این خط وزنهای مدل را با استفاده از الگوریتم بهینهساز (optimizer) که قبلاً تعریف شده است، بروزرسانی میکند. بهینهساز گام مناسب را برای تغییر وزنهای مدل به سمت کمینه خطا پیدا میکند

• o = epoch + 1: این خط شمارنده دوره آموزش را با ۱ افزایش میدهد. این کار برای نمایش بهتر نتایج در خروجی صورت میگیرد.

• epochs.append(o): این خط شمارنده دوره آموزش را به یک لیست به نام epochs اضافه میکند. این لیست برای رسم نمودار خطا در برابر دوره آموزش مورد استفاده قرار میگیرد.

• epoch_losses.append(loss.item()): این خط مقدار خطا را به یک لیست به نام epoch_losses اضافه میکند. این لیست هم برای رسم نمودار خطا در برابر دوره آموزش مورد استفاده قرار میگیرد.

• print(f'Epoch {o}/{num_epochs}'): این خط شماره دوره آموزش را در خروجی چاپ میکند. برای مثال، اگر دوره آموزش سوم باشد و تعداد کل دورههای آموزشی ۱۰ باشد، این خط Epoch 3/10 را چاپ میکند.

• print(f"Loss: {loss.item()}"): این خط مقدار خطا را در خروجی چاپ میکند. برای مثال، اگر خطا برابر با ۰٫۵ باشد، این خط Loss: 0.5 را چاپ میکند.

رسم نمودار

در این کد، دارید نمودار خطا (loss) در برابر دوره آموزشی (epoch) را با استفاده از کتابخانه matplotlib رسم میکنید. نمودار خطا نشان میدهد که چگونه خطا در طول آموزش شبکه عصبی تغییر میکند. هر چه خطا کمتر باشد، یعنی شبکه عصبی بهتر عمل کرده است. در اینجا، مراحل کد را توضیح میدهم:

• plt.plot(epochs, epoch_losses): این خط نقاط نمودار را با استفاده از دو لیست epochs و epoch_losses تعیین میکند. لیست epochs شامل شماره دورههای آموزشی است و لیست epoch_losses شامل مقدار خطا در هر دوره آموزشی است. این خط با وصل کردن نقاط با خطوط، یک نمودار خطی رسم میکند.

• plt.title('Training Loss Plot'): این خط عنوان نمودار را با استفاده از رشته 'Training Loss Plot' تعیین می کند.

- plt.xlabel('Epoch axis'): این خط برچسب محور افقی (x) را با استفاده از رشته 'Epoch axis' تعیین میکند
- plt.ylabel('Loss axis'): این خط برچسب محور عمودی (y) را با استفاده از رشته 'Loss axis' تعیین میکند
- plt.show(): این خط نمودار را نشان میدهد

Sara Sadat Younesi

IMPORT LIBRARY FROM TORCH

```
import torch
import torch.nn as nn
from PIL import Image
import numpy as np
import torch.optim as optim
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
```

Read image and size of image

```
image1 = Image.open('Q5_1.png')
image2 = Image.open('Q5_2.png')
image3 = Image.open('Q5_3.png')
```

Check type of image

```
print(type(image1))
```

[3]

```
<class 'PIL.PngImagePlugin.PngImageFile'>
```

```
transform = transforms.Compose([transforms.Resize((8, 8)), tr
tensor_image1 = transform(image1)
tensor_image2 = transform(image2)
tensor_image3 = transform(image3)

print(tensor_image1)
print(tensor_image2)
print(tensor_image3)

y = torch.tensor([0, 1, 2])
```

[4]

```
tensor([[[0.9098, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980,
         [0.9059, 0.8980, 0.8980, 0.8980, 0.8863, 0.8353, 0.8353,
         [0.9059, 0.8863, 0.8431, 0.8784, 0.7843, 0.4627, 0.4667,
         [0.8471, 0.8196, 0.4667, 0.7098, 0.4510, 0.6980, 0.6980,
```

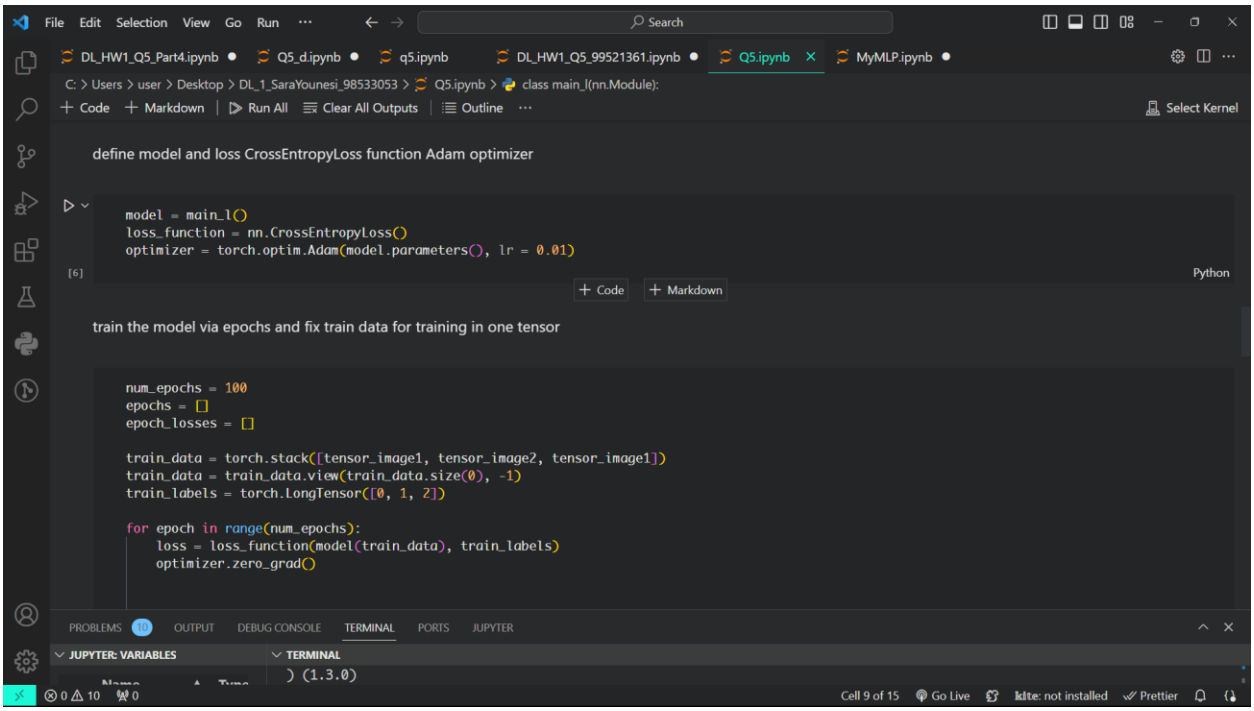
```
tensor([[[0.9098, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.9020],
         [0.9059, 0.8980, 0.8980, 0.8980, 0.8863, 0.8353, 0.8353, 0.8902],
         [0.9059, 0.8863, 0.8431, 0.8784, 0.7843, 0.4627, 0.4667, 0.7725],
         [0.8471, 0.8196, 0.4667, 0.7098, 0.4510, 0.6980, 0.6980, 0.4000],
         [0.4275, 0.7569, 0.3569, 0.3882, 0.3569, 0.4078, 0.4078, 0.3529],
         [0.3451, 0.6941, 0.4000, 0.7529, 0.8157, 0.8157, 0.8157, 0.8196],
         [0.3529, 0.3961, 0.4157, 0.8314, 0.8980, 0.8980, 0.8980, 0.9020],
         [0.8314, 0.8196, 0.8275, 0.8980, 0.9059, 0.9059, 0.9059, 0.9098]]])
tensor([[[0.9059, 0.9059, 0.9059, 0.8980, 0.7804, 0.4706, 0.7098, 0.4314],
         [0.8980, 0.8980, 0.8980, 0.8431, 0.5176, 0.6706, 0.5059, 0.7765],
         [0.8275, 0.8863, 0.8941, 0.8784, 0.7294, 0.5020, 0.7804, 0.8980],
         [0.4000, 0.7608, 0.8196, 0.7647, 0.4549, 0.7686, 0.8863, 0.9098],
         [0.3529, 0.4078, 0.4157, 0.4078, 0.4157, 0.8275, 0.8980, 0.9098],
         [0.8235, 0.8235, 0.8235, 0.8235, 0.8314, 0.8863, 0.8980, 0.9098],
         [0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.9098],
         [0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.9137]]])
tensor([[[0.9255, 0.9137, 0.9137, 0.9137, 0.9137, 0.9137, 0.9137, 0.9137],
         [0.9059, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980, 0.8980],
         [0.9059, 0.8863, 0.8353, 0.8863, 0.8941, 0.8941, 0.8863, 0.8235],
         [0.9059, 0.8235, 0.4588, 0.7608, 0.8157, 0.8157, 0.7529, 0.3961],
         [0.9059, 0.8275, 0.4196, 0.4118, 0.4157, 0.4157, 0.4078, 0.3569],
         [0.9059, 0.8902, 0.8353, 0.8196, 0.7686, 0.8196, 0.8275, 0.8275],
         [0.9059, 0.8980, 0.8980, 0.8392, 0.5294, 0.8431, 0.8980, 0.8980],
         [0.9020, 0.8902, 0.8902, 0.8784, 0.8314, 0.8824, 0.8902, 0.8902]]])
```

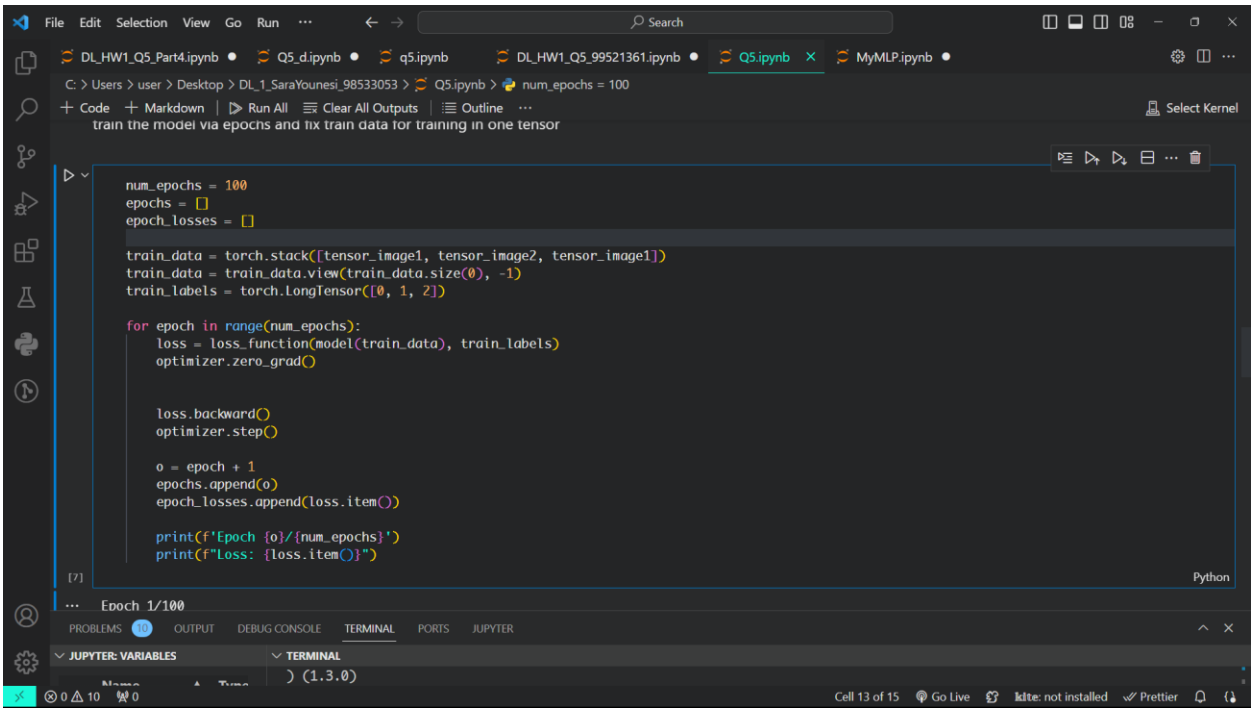
Define Base MLP Architecture

Define input layer with 64 neroun Define hidden layer with 16 noren Define output layer with 3 noren Define Activation function AND forward in network step by step

```python
class main_l(nn.Module):
    def __init__(self):
        super(main_l, self).__init__()

        self.input = nn.Linear(64, 16)
        self.hidden = nn.Linear(16, 16)
        self.output = nn.Linear(16, 3)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax()


    def forward(self, x):
        x = self.input(x)
        x = self.hidden(x)
        x = self.relu(x)
        x = self.output(x)
        x = self.softmax(x)

        return x
```

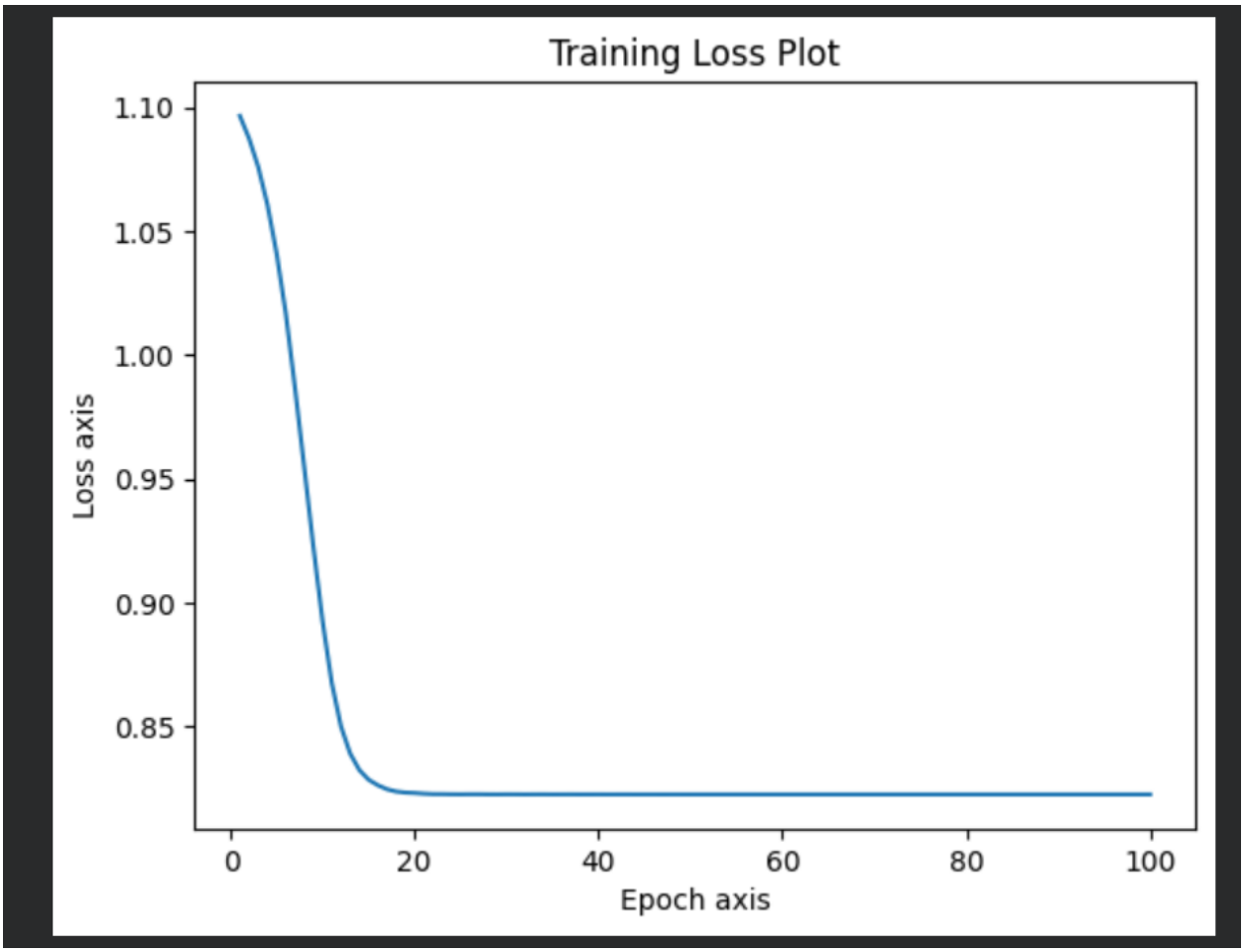
Python


File Edit Selection View Go Run
Search
DL_HW1_Q5_Part4.ipynb
Q5_d.ipynb
q5.ipynb
DL_HW1_Q5_99521361.ipynb
Q5.ipynb
MyMLP.ipynb
C: > Users > user > Desktop > DL_1_SaraYounesi_98533053 > Q5.ipynb > class main_l(nn.Module):
Code Markdown Run All Clear All Outputs Outline
Select Kernel
define model and loss CrossEntropyLoss function Adam optimizer
model = main_l()
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)
[6]
Python
Code Markdown
train the model via epochs and fix train data for training in one tensor
num_epochs = 100
epochs = []
epoch_losses = []
train_data = torch.stack([tensor_image1, tensor_image2, tensor_image1])
train_data = train_data.view(train_data.size(0), -1)
train_labels = torch.LongTensor([0, 1, 2])
for epoch in range(num_epochs):
loss = loss_function(model(train_data), train_labels)
optimizer.zero_grad()
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
JUPYTER: VARIABLES
TERMINAL
) (1.3.0)
Cell 9 of 15 Go Live kite: not installed Prettier


File Edit Selection View Go Run
Search
DL_HW1_Q5_Part4.ipynb
Q5_d.ipynb
q5.ipynb
DL_HW1_Q5_99521361.ipynb
Q5.ipynb
MyMLP.ipynb
C: > Users > user > Desktop > DL_1_SaraYounesi_98533053 > Q5.ipynb > num_epochs = 100
Code Markdown Run All Clear All Outputs Outline
Select Kernel
train the model via epochs and fix train data for training in one tensor
num_epochs = 100
epochs = []
epoch_losses = []
train_data = torch.stack([tensor_image1, tensor_image2, tensor_image1])
train_data = train_data.view(train_data.size(0), -1)
train_labels = torch.LongTensor([0, 1, 2])
for epoch in range(num_epochs):
loss = loss_function(model(train_data), train_labels)
optimizer.zero_grad()
loss.backward()
optimizer.step()
o = epoch + 1
epochs.append(o)
epoch_losses.append(loss.item())
print(f'Epoch {o}/{num_epochs}')
print(f"Loss: {loss.item()}")
[7]
Python
Epoch 1/100
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
JUPYTER: VARIABLES
TERMINAL
) (1.3.0)
Cell 13 of 15 Go Live kite: not installed Prettier

```
Epoch 1/100
Loss: 1.0965622663497925
Epoch 2/100
Loss: 1.087426781654358
Epoch 3/100
Loss: 1.0758947134017944
Epoch 4/100
Loss: 1.0606800317764282
Epoch 5/100
Loss: 1.0409187078475952
Epoch 6/100
Loss: 1.016611933708191
Epoch 7/100
Loss: 0.9865768551826477
Epoch 8/100
Loss: 0.9546770453453064
Epoch 9/100
Loss: 0.922368049621582
Epoch 10/100
Loss: 0.8923851847648621
Epoch 11/100
Loss: 0.8674263954162598
Epoch 12/100
Loss: 0.8499452471733093
Epoch 13/100
...
```

```
    Loss: 0.8224970698356628
    Epoch 63/100
    Loss: 0.822497546672821
    Epoch 64/100
    Loss: 0.8224976658821106
    Epoch 65/100
    Loss: 0.8224965929985046
    Epoch 66/100
    Loss: 0.8224977850914001
    Epoch 67/100
    Loss: 0.8224965929985046
    Epoch 68/100
    Loss: 0.8224971890449524
    Epoch 69/100
    Loss: 0.8224968910217285
    Epoch 70/100
    Loss: 0.8224966526031494
    Epoch 71/100
    Loss: 0.8224970698356628
    Epoch 72/100
    Loss: 0.8224963545799255
    Epoch 73/100
    ...
    Epoch 99/100
    Loss: 0.8224959373474121
    Epoch 100/100
    Loss: 0.8224959373474121
```

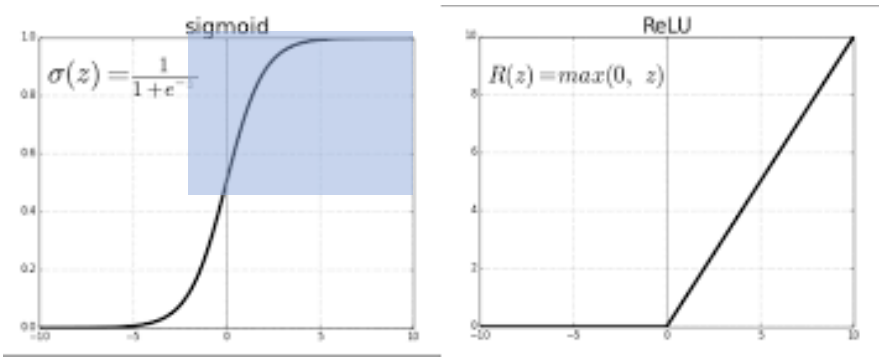


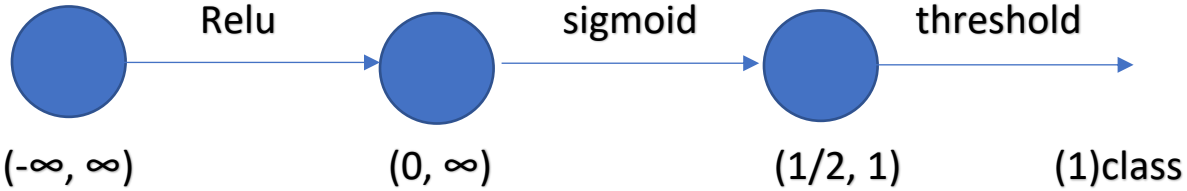
۶- یک شبکه عصبی چندلایه(MLP) را در نظر بگیرید که برای دسته بندی دو کلاس مورد استفاده قرار می گیرد. خروجی نورون آخر را z در نظر بگیرید و خروجی شبکه عصبی به صورت زیر محاسبه می شود:

$$y = \sigma(RELU(z))$$

که در آن y خروجی پیش بینی شده است، σتابع فعال ساز سیگموئید و RELU نمایانگر تابع فعال ساز واحد خطی بازگشتی است MLP .از یک آستانه .۵ ۰ استفاده می کند و خروجی های بزرگتر یا مساوی .۵ ۰ را به عنوان کلاس ۱ در نظر می گیرد و خروجی های کمتر از .۵ ۰ را به عنوان کلاس ۰ دسته بندی می کند .در استفاده از این شبکه عصبی برای دسته بندی دو کلاسه در نظر گرفته شده، چه مشکلات یا چالش هایی با توجه به ترکیب توابع فعال ساز و آستانه .۵ ۰ ممکن است وجود داشته باشد؟(۲۰ نمره)

تحلیل توابع فعال ساز و آستانه مورد نظر)





باتوجه به ترتیب توابع و آنچه که مشاهده می کنیم خروجی تابع غیرخطی رلو همواره مثبت خواهد بود و خروجی های منفی را تاثیرشان را حذف می کند سپس انچه که خواهیم داشت اعدادی از بازه ی صفر تا مثبت بی نهایت خواهد بود که با توجه به برد تابع سیگموید این خروجی ½ تا ۱ را همواره به ما می دهد و با توجه به حد آستانه ی مورد نظر ما همواره خروجی کلاس یک را خواهیم داشت !

و کلاس صفر هیچ گاه در پیش بینی های ما دیده نمیشوند که به دلیل نورون های مرده این اتفاق می افتد.

دلیل اصلی «نرون‌های مرده» این است که نورون‌ها در موقعیتی قرار می‌گیرند که همیشه ارزش خاصی تولید می‌کنند و گرادیان صفر دارند .این وضعیت بیشتر با ReLU همراه است. در نتیجه برخی از نورون ها می میرند، فعال سازی غیرقابل تغییر ایجاد می کنند و هرگز احیا نمی شوند.

اگر با مشکل مرگ نورون در شبکه مواجه هستیم، تابع Leaky ReLU می‌تواند گزینه‌ی بسیار خوبی باشد.

.، به‌دلیل مشکل مرگ نورون، کمتر استفاده می‌شود(Tanh) تابع تانژانت هایپربولیک

یکی از مشکلات احتمالی که ممکن است در صورت استفاده از تابع فعال سازی ReLU و به دنبال آن تابع سیگموئید در یک شبکه عصبی چند لایه و حد آستانه ما ۲/۱ باشد، مشکل نورون مرده است .این زمانی است که برخی از نورون های شبکه به هر ورودی پاسخ نمی دهند و خروجی صفر تولید می کنند و آنها را برای یادگیری بی فایده می کند .تابع فعال‌سازی ReLU حداکثر مقدار

بین صفر و مقدار ورودی را خروجی می‌دهدتابع سیگموئید مقداری بین ۰ و ۱ را خروجی می دهد و با صفر شدن مقدار ورودی به ۰,۵ نزدیک می شود .اگر حد آستانه ۲/۱ داشته باشیم، به این معنی است که هر مقدار ورودی که کمتر یا مساوی صفر باشد توسط تابع سیگموئید به ۰,۵ نگاشت می شود و هر مقدار ورودی که بزرگتر از صفر باشد به یک مقدار نگاشت می شود .بالای ۰,۵ توسط تابع سیگموئید .اگر از این دو تابع در یک شبکه عمیق استفاده کنیم، ممکن است به تعدادی نورون برسیم که همیشه ورودی های منفی یا صفر را از لایه قبلی دریافت می کنند و بنابراین پس از اعمال تابع ReLU همیشه خروجی صفر دارند .این نورون ها پس از اعمال تابع سیگموئید، بدون توجه به ورودی، همیشه خروجی ۰,۵ خواهند داشت .این بدان معناست که این نورون‌ها به فرآیند یادگیری کمک نمی‌کنند و توسط الگوریتم پس انتشار به‌روزرسانی نمی‌شوند .این نورون ها نورون های مرده نامیده می شوند و ظرفیت موثر شبکه را کاهش می دهند .یکی از راه‌های جلوگیری از این مشکل، استفاده از یک اصطلاح بایاس مثبت کوچک برای هر نورون است، مانند ۰,۰۱، که می‌تواند از صفر یا منفی شدن ورودی‌ها جلوگیری کند .راه دیگر استفاده از توابع فعال سازی متفاوتی است که خروجی صفر ندارند، مانندReLU ، ELUیا tanh که دارای نشتی هستند

یکی از مشکلات احتمالی که ممکن است در صورت استفاده از تابع ReLU به دنبال تابع سیگموئید در یک شبکه عصبی چند لایه با آن مواجه شویم، مشکل گرادیان ناپدید شدن است .این زمانی است که گرادیان های تابع از دست دادن نسبت به وزن ها بسیار کوچک یا صفر می شود و شبکه را قادر به یادگیری موثر نمی کند .تابع ReLU دارای مشتق ۱ برای ورودی های مثبت و ۰ برای ورودی های منفی است .تابع سیگموئید مشتقی دارد که همیشه بین ۰ و ۱ است و با بزرگ شدن یا بسیار کوچک شدن ورودی به ۰ نزدیک می شود .اگر این دو تابع را در یک شبکه عمیق ترکیب کنیم، ممکن است به لایه‌های زیادی برسیم که دارای گرادیان صفر یا نزدیک به صفر هستند، به خصوص اگر ورودی‌ها منفی یا اشباع شده باشند .این بدان معناست که وزن های موجود در این لایه ها زیاد یا اصلا به روز نمی شوند و شبکه قادر به یادگیری ویژگی های پیچیده نخواهد بود.

از ترکیب این دو مشکل vanishing gradient وexploding gradiant

محو شدگی گرادیان ، مشکلی است که حین شبکه های عصبی مصنوعی با استفاده از روش های یادگیری مبتنی بر گرادیان رخ میدهد. در این نوع روشها به منظور بروزرسانی پارامترهای شبکه عصبی از گرادیان استفاده میشود. هر پارامتر با توجه به میزان اثری که در نتیجه نهایی شبکه داشته است مورد تغییر قرار میگیرد. این مهم با استفاده از مشتق جزئی تابع خطا نسبت به هر پارامتر در هر تکرار فرایند آموزش صورت میپذیرد. مشکل محو شدگی اشاره به این مساله دارد مقادیر گرادیان ها با حرکت به سمت ابتدای شبکه رفته رفته به حدی کوچک میشوند که تغییرات وزن بصورت ناچیزی صورت میگیرد و به این علت فرایند آموزش بشدت کند میشود

قطه‌ی مقابل محوشدگی گرادیان، مشکل انفجار گرادیان یا همان exploding gradients است که به جای اضمحلال و محوشدن گرادیان، ممکن است آن را بیش از اندازه بزرگ نماید و به خاطر همین الگوریتم نتواند به یک همگرایی (converge) در میان وزن‌ها دست پیدا کند.

در شبکه‌های عصبی عمیق با زیاد شدن تعداد لایه‌ها، بایستی هر کدام از وزن‌ها در لایه‌های مختلف با دقت بیشتری آپدیت شوند. اگر این گونه نشد، ممکن است وزن‌ها بیش از مقدار مورد انتظار، آپدیت شده و در اصطلاح الگوریتم به جای همگرایی به یک جواب بهینه (ترکیب وزن‌های مناسب)، واگرا (diverge) شده و نتواند به جواب‌های بهینه دست پیدا کند.

شیب ناپدید شدن یکی از بزرگترین چالش ها هنگام آموزش شبکه عصبی عمیق است .این وضعیتی است که در آن یک شبکه عصبی عمیق قادر به انتشار مجدد گرادیان از لایه خروجی به اولین لایه پنهان نیست .اغلب زمانی اتفاق می‌افتد که سعی می‌کنید یک شبکه عصبی عمیق با یک تابع فعال‌سازی سیگموئید روی لایه‌های پنهان آن بسازید.

مشکل مرگ نورون یا مرگ ReLU دارد؛ یعنی زمانی‌که ورودی صفر یا نزدیک به صفر باشد، تابع ReLU دیگر عملکردی ندارد و به‌بیان دیگر، می‌میرد. در این صورت، مقدار گرادیان تابع صفر می‌شود و شبکه نمی‌تواند عملیات پس انتشار را انجام دهد و آموزش ببیند.

اینکه آستانه هم یک عدد ثابت قرار دارد می تواند مشکل ساز شودد و تبدیل به پیش بینی غلط شود در موارد غیر خطی و توانایی کلاس بندی را کاهس دهد

در نهایت نتوان الگو ها راا به درستی شناسایی کند و و کاهش در دقت کلاس بندی شود و و

۷- تحلیل خود برای سوالات زیر را بنویسید(۲۰ نمره)

.الف) به نظر شما مهم ترین تفاوت یادگیری ماشین و یادگیری عمیق در چیست؟

ب) فرض کنید یک شبکه یادگیری عمیق دارای ۱۶ لایه است. به نظر شما لایه ۷ ام برای دستیابی به نتیجه نهایی در طبقه بندی مناسب تر است یا لایه ۱۱ ام؟ چرا؟

ج) به نظر شما برای تقریب توابع استفاده از شبکه های عمیق تر کاراتر است یا شبکه های عریض تر؟ چرا؟

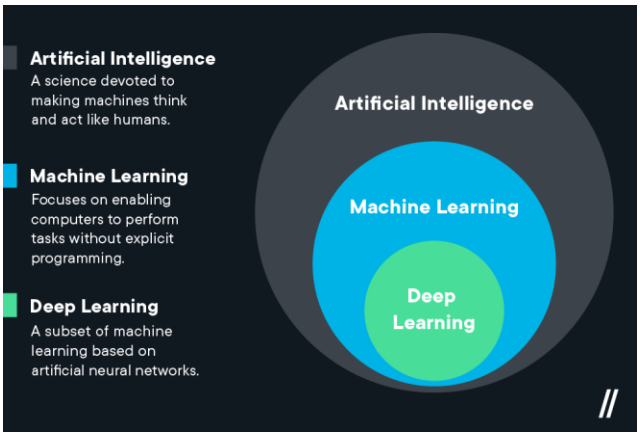
د)مزایا و معایب افزودن لایه های بیشتر به شبکه عصبی عمیق چیست؟

قسمت الف)

۱. مهم ترین تفاوت آن استفاده از شبکه های عصبی عمیق است. در یادگیری عمیق شبکه های عصبی با تعداد بسیار زیادی از لایه ها عموما بیش تر از سه لایه برای حل مسائل پیچیده استفاده می شونددر حالی که در یادگیری ماشین از الگوریتم های مختلف آماری و یادگیری ماشین سنتی استفاده می شود و هدف از یادگیری عمیق بهبود عملکرد و دقت در حل مسائل پیچیده است.
۲. در ویژگی عمیق ویژگی ها به طور خودکار کسف می شوند نه نه دستی

۳. یادگیری ماشینی به معنای یادگیری کامپیوترها از داده ها با استفاده از الگوریتم ها برای انجام یک کار بدون برنامه ریزی صریح است .یادگیری عمیق از ساختار پیچیده ای از الگوریتم های مدل سازی شده بر روی مغز انسان استفاده می کند . این امکان پردازش داده های بدون ساختار مانند اسناد، تصاویر و متن را فراهم می کند.

۴. به نظر من مهم ترین تفاوت یادگیری ماشین و یادگیری عمیق در ساختار و عمق الگوریتم های آن ها است. یادگیری ماشین از الگوریتم های ساده تر و کم لایه تری استفاده می کند که برای حل مسائل خاص و با داده های کمتر مناسب هستند. اما یادگیری عمیق از شبکه های عصبی مصنوعی پیچیده و چند لایه ای استفاده می کند که برای حل مسائل پیچیده و با داده های زیاد مناسب هستند. شبکه های عصبی مصنوعی قابلیت یادگیری و تفسیر ویژگی های سطح بالا و انتزاعی را دارند که برای تشخیص الگوها، پردازش زبان طبیعی، تشخیص تصویر و صوت و غیره کاربرد دارند.



MACHINE LEARNING VS DEEP LEARNING

	MACHINE LEARNING	DEEP LEARNING
Approach	Requires structure data	Does not require structure data
Human Intervention	Requires human intervention for mistakes	Does not require human intervention for mistakes
Hardware	Can function on CPU	Requires GPU / significant computing power
Time	Takes seconds to hours	Takes weeks
Uses	Forecasting, predicting and other simple applications	More complex applications like autonomous vehicles

قسمت ب)

به نظر من لایه ۱۱ ام برای دستیابی به نتیجه نهایی در طبقه بندی مناسب تر است. چون لایه های بالاتر شبکه عصبی عمیق معمولاً ویژگی های پیچیده تر و انتزاعی تر را یاد می گیرند که برای تشخیص الگوها و دسته بندی داده ها مفیدتر هستند. لایه های پایین تر شبکه عصبی عمیق معمولاً ویژگی های ساده تر و محلی را یاد می گیرند که برای بازسازی تصویر و کاهش نویز مفیدتر هستند. بنابراین، لایه ۱۱ ام که در نزدیکی لایه خروجی قرار دارد، احتمالاً دارای اطلاعات بیشتر و کارآمدتری برای طبقه بندی است. البته این پاسخ به نوع شبکه عصبی عمیق، تابع فعال سازی، تعداد نورون ها و داده های ورودی نیز بستگی دارد.زیرا در برخی از مسائل لایه های ابتدایی به عنوان ویژگی های اولیه مهم هستند و در برخی مسائل لایه های انتهایی برای تصمیم نهایی مهم هستند اما در حالت کلی جواب لایه ی ۱۱ می باشد.

قسمت ج)

به طور عمومی شبکه های عمیق تر

۱. یک شبکه عصبی به اندازه کافی گسترده با تنها یک لایه پنهان می تواند هر تابع (معقول) را با توجه به داده های آموزشی کافی تقریبی کند .با این حال، با استفاده از یک شبکه بسیار گسترده و کم عمق، چند مشکل وجود دارد . مسئله اصلی این است که این **شبکه های بسیار گسترده و کم عمق در به خاطر سپردن بسیار خوب هستند، اما در تعمیم آنچنان خوب نیستند** .بنابراین، اگر شبکه را با هر مقدار ورودی ممکن آموزش دهید، یک شبکه فوق گسترده در نهایت می تواند مقدار خروجی مربوطه را که می خواهید به خاطر بسپارد .اما این مفید نیست زیرا برای هر برنامه کاربردی عملی شما تمام مقدار ورودی ممکن را برای آموزش نخواهید داشت .**مزیت لایه های متعدد این است که می توانند ویژگی ها را در سطوح مختلف انتزاع بیاموزند**.

۲. **لایه های چندگانه در تعمیم بسیار بهتر هستند زیرا آنها تمام ویژگی های میانی بین داده های خام و طبقه بندی سطح بالا را یاد می گیرند** .بنابراین این توضیح می دهد که چرا ممکن است از یک شبکه عمیق به جای یک شبکه بسیار گسترده اما کم عمق استفاده کنید می‌خواهیم شبکه‌تان تا حد امکان کوچک باشد تا نتایج خوبی حاصل شود .همانطور که اندازه شبکه را افزایش می دهید، در واقع فقط پارامترهای بیشتری را معرفی می کنید که شبکه شما باید یاد بگیرد، و **در نتیجه شانس بیش از حد برازش** را افزایش می دهید .اگر یک شبکه بسیار گسترده و بسیار عمیق بسازید، این شانس را دارید که هر لایه فقط آنچه را که خروجی می خواهید به خاطر بسپارد، و در نهایت با یک شبکه عصبی مواجه می شوید که نمی تواند به داده های جدید تعمیم یابد .جدای از **شبح بیش از حد برازش، هرچه شبکه شما گسترده تر باشد، آموزش آن بیشتر طول می کشد** .آموزش شبکه‌های عمیق می‌تواند از نظر محاسباتی بسیار پرهزینه باشد، بنابراین انگیزه‌ای قوی وجود دارد که آنها را به اندازه‌ای گسترده کنیم که به خوبی کار کنند، اما نه گسترده‌تر.

۳. به نظر من برای تقریب توابع، استفاده از شبکه های عمیق تر یا عریض تر بستگی به نوع تابع و میزان پیچیدگی آن دارد. به طور کلی، می توان گفت **که شبکه های عمیق تر قادر به یادگیری ویژگی های انتزاعی تر و پیچیده تر هستند** که برای تقریب توابع غیرخطی و غیرمحدب مناسب تر هستند. اما شبکه های عریض تر قادر به یادگیری ویژگی های سطح پایین تر و ساده تر هستند که برای تقریب توابع خطی و محدب مناسب تر هستند. البته این موضوع به شروط دیگری نظیر اندازه داده، نوع فعالساز، روش بهینه سازی و غیره نیز بستگی دارد.

۴.

We show that, for a large class of piecewise smooth functions, the number of neurons needed by a shallow network to approximate a function is exponentially larger than the corresponding number of neurons needed by a deep network for a given degree of function approximation. First, we consider univariate functions on a bounded interval and require a neural network to achieve an approximation error of ε uniformly over the interval. We show that shallow networks (i.e., networks whose depth does not depend on ε) require Ω(poly(1/ε)) neurons while deep networks (i.e., networks whose depth grows with 1/ε) require O(polylog(1/ε)) neurons. We then extend these results to certain classes of important multivariate functions. Our results are derived for neural networks which use a combination of rectifier linear units (ReLUs) and binary step units, two of the most popular type of activation functions. Our analysis builds on a simple observation: the multiplication of two bits can be represented by a ReLU.

قسمت د)

مزایا :

۱. افزایش قدرت تمایزگری و تشخیص الگوها: هر چه لایه های بیشتری استفاده شوند، شبکه عصبی قادر خواهد بود ویژگی های پیچیده تر و انتزاعی تر را یاد بگیرد و بهتر با داده های ناهمگن کنار بیاید.

۲. افزایش دقت کلی: هر چه لایه های بیشتری استفاده شوند، دقت کلی بالاتر می رود؛ زیرا شبکه عصبی می تواند روابط پنهان و غیرخطی بین داده ها را کشف کند

۳. پیش بینی بهتر و کار کردن بهتر با ورودی های بزرگ.

۴. تعداد لایه های پنهان راضی شده استفاده شود، نتایج بهتری با پیچیدگی زمانی بسیار کمتر به دست می آوریم .از طرفی اگر تعداد لایه های پنهان را افزایش دهیم تا حد زیادی می توان به دقت مناسبی دست یافت اما شبکه عصبی نسبت به روش های قبلی پیچیده تر می شود.

معایب :

۱. افزایش زمان آموزش و پیچیدگی محاسبات: هر چه لایه های بیشتری استفاده شوند، شبکه عصبی کندتر اجرا می شود، زیرا تعداد محاسبات افزایش می یابد. این مسئله ممکن است منجر به نیاز به منابع سخت افزاری بالاتر و گران تر شود.

۲. افزایش خطر بیش برازش: هر چه لایه های بیشتری استفاده شوند، شبکه عصبی پیچیده تر می شود و ممکن است به جای یادگیری قوانین کلی، جزئیات نامربوط و خاص داده های آموزش را یاد بگیرد. این مسئله باعث می شود که شبکه عصبی روی داده های جدید عملکرد خوب نداشته باشد

۳. Without a large training set, an increasingly large network is likely to overfit and in turn reduce accuracy on the test data.

منابع استفاده شده :

توابع فعالساز (Activation Functions) چیست و چه کاربردهایی دارد؟ | وبلاگ کافه تدریس(cafetadris.com)

https://hooshio.com/%D8%AA%D9%88%D8%A7%D8%A8%D8%B9-%D9%81%D8%B9%D8%A7%D9%84-%D8%B3%D8%A7%D8%B2%DB%8C

سرچ در GPT و Bing با ترامپت متن سوالها

machine learning - Why are neural networks becoming deeper, but not wider? - Cross Validated (stackexchange.com)

[1610.04161] Why Deep Neural Networks for Function Approximation? (arxiv.org)

Probit model - Wikipedia

Module 4 - Logistic Regression | The Programming Foundation