

به نام خدا

تمرين سري اول
درس مبانی يادگيری عميق
دکتر مرضیه داود آبادی

فرزان رحمانی
۹۹۵۲۱۲۷۱

سوال اول
(الف)

$C \in \{0, 1\}$ $\text{Words} \in \{\text{اجنبی}, \text{علی}, \text{فرمتنی}, \text{فنادری}\}$

$$P(C | \text{Words}) = \frac{P(C) P(\text{Words} | C)}{P(\text{Words})} \quad (1)$$

محرج کسر برای این دو کلاس نیسان است و نیازی به محاسبه آن نیست.

$$P(C | \text{Words}) \approx P(C) P(\text{Words} | C) = P(C) \prod_{i=1}^n P(\text{Words}_i | C)$$

$$P(C=0) = \frac{2}{4} = \frac{1}{2} \quad P(C=1) = \frac{2}{4} = \frac{1}{2}$$

احتمال انتخاب کلاس ۰ احتمال انتخاب کلاس ۱

$$P(\text{Words} | C=0) = \prod_{i=1}^n P(\text{Words}_i | C=0) = P(0|\text{اجنبی}).P(0|\text{علی}).P(0|\text{فرمتنی}).P(0|\text{فنادری})$$

$$P(\text{Words} | C=1) = \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \Rightarrow P(C=0 | \text{Words}) = \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4}$$

$$P(\text{Words} | C=1) = \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \Rightarrow P(C=1 | \text{Words}) = \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4}$$

$$P(C=0 | \text{Words}) = \frac{2^4}{4^4} \approx 4,1 \times 10^{-4}, \quad P(C=1 | \text{Words}) = \frac{1}{4^4} \approx 2,4 \times 10^{-4}$$

احتمال کلاس ۰ بیشتر از احتمال کلاس ۱ است پس نمونه سنت در کلاس ۰ مغلوب دارد.

یک مثال کارمزد محاسبه $P(\text{Words}_i | C)$ را در زیر آورده ایم بقید هم به همین روش اسما.

$$P(\text{Words}_3 | C=1) = \frac{\text{تعداد کلمه علی در کلاس ۱}}{\text{تعداد کلمات در کلاس ۱}} = \frac{1}{1} = 1$$

$$P(\text{Words}_3 | C=1) = \frac{\text{freq}(cde, C=1)}{N_{C=1}} = \frac{1}{8} = 0.125$$

Frequency of words in class 1

Laplace Smoothing

Laplace smoothing is a smoothing technique that handles the problem of zero probability in Naïve Bayes. Using Laplace smoothing, we can represent $P(w'|positive)$ as

$$P(w'|positive) = \frac{\text{number of reviews with } w' \text{ and } y = \text{positive} + \alpha}{N + \alpha * K}$$

Here,

alpha represents the smoothing parameter,

K represents the number of dimensions (features) in the data, and

N represents the number of reviews with $y=positive$

If we choose a value of $\alpha \neq 0$ (not equal to 0), the probability will no longer be zero even if a word is not present in the training dataset.

Interpretation of changing alpha

Let's say the occurrence of word w is 3 with y=positive in training data. Assuming we have 2 features in our dataset, i.e., K=2 and N=100 (total number of positive reviews).

$$P(w|positive) = \frac{3 + \alpha}{100 + 2 * \alpha}$$

Case 1- when alpha=1

$$P(w'|positive) = 3/102$$

Case 2- when alpha = 100

$$P(w'|positive) = 103/300$$

Case 3- when alpha=1000

$$P(w'|positive) = 1003/2100$$

As alpha increases, the likelihood probability moves towards uniform distribution (0.5). Most of the time, alpha = 1 is being used to remove the problem of zero probability.

Conclusion

Laplace smoothing is a smoothing technique that helps tackle the problem of zero probability in the Naïve Bayes machine learning algorithm. Using higher alpha values will push the likelihood towards a value of 0.5, i.e., the probability of a word equal to 0.5 for both the positive and negative reviews. Since we are not getting much information from that, it is not preferable. Therefore, it is preferred to use alpha=1.

$$P(C=0) = P(C=1) = \frac{2}{4} = \frac{1}{2} = 0.5$$

ب) همان طرکه در توصیفات بالا آمده است باید از همارسازی لابلس

$$\prod_{i=1}^n P(\text{Words}_i | C) = 0 \quad \text{استفاده کنیم چونه آن را نگذیم}$$

$$\text{صفری شود، لعلی چون } P(C | \text{Words}) = P(\text{Words} | C)$$

Laplace Smoothing

$$P(\text{Words}_i | C) = \frac{\text{freq}(\text{Words}_i, C) + \alpha}{N_C + \alpha \cdot K} \quad \alpha = 1 \quad \text{freq}(\text{Words}_i, C) + 1$$

$$\left\{ \begin{array}{l} P(\text{Words}_i | C=0) = \frac{\text{freq}(\text{Words}_i, C=0) + 1}{9 + 7} \\ P(\text{Words}_i | C=1) = \frac{\text{freq}(\text{Words}_i, C=1) + 1}{8 + 7} \end{array} \right.$$

$\frac{N_C + K}{\text{num. of words in class} + \text{num. of words in data}}$
 $K = 7$
 تعداد کلمات متمایز در
 واژه‌اندازه‌ها هشت
 است: {تاریخ، فرهنگی،
 علمی، اقتصادی، اجتماعی، سیاسی، ورزشی}

$$P(C | \text{Words}) = \frac{P(C) P(\text{Words} | C)}{P(\text{Words})} \approx \frac{P(C) P(\text{Words} | C)}{P(C=0) + P(C=1)} = \frac{1}{2} P(\text{Words} | C) \quad P(C=0) = P(C=1) = 0.5$$

$$P(\text{Words} | C=0) = \prod_{i=1}^n P(\text{Words}_i | C=0) = \frac{2+1}{9+7} \times \frac{3}{16} \times \frac{3}{16} \times \frac{2}{16} \times \frac{1}{16}$$

$$P(\text{Words} | C=1) = P(\text{تاریخ} | 0) P(\text{فرهنگی} | 0) P(\text{علمی} | 0) P(\text{اقتصادی} | 0) P(\text{اجتماعی} | 0) P(\text{سیاسی} | 0) P(\text{ورزشی} | 0)$$

$$= \frac{1+1}{8+7} \times \frac{2}{15} \times \frac{2}{15} \times \frac{3}{15} \times \frac{1}{15}$$

$$P(C=0 | \text{Words}) = \frac{1}{2} \times \frac{2^3 \times 2^3}{14^5} = \frac{2^3 \times 2^3}{14^5}, \quad P(C=1 | \text{Words}) = \frac{1}{2} \times \frac{2^3 \times 3^3}{14^5} = \frac{2^3 \times 3^3}{14^5}$$

$$P(C=0 | \text{Words}) \approx 2,0 \times 10^{-5} \quad \text{پیشتر}, \quad P(C=1 | \text{Words}) \approx 1,9 \times 10^{-5}$$

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

مراجع:

<https://towardsdatascience.com/laplace-smoothing-in-na%AFve-bayes-algorithm-9c237a8bdece>

سوال سوم

این سوال شبیه مطالب اسلاید زیر است:



- فرض کنید مشاهدات از یک مدل خطی همراه با نویز نرمال (گاوی) جمع آوری شده‌اند

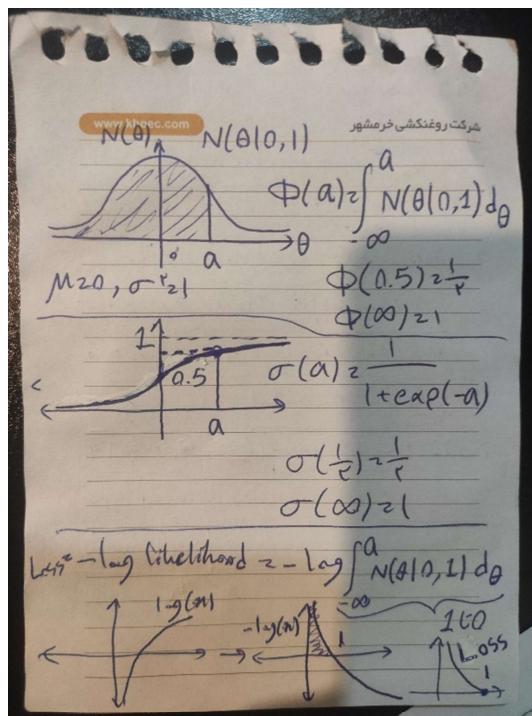
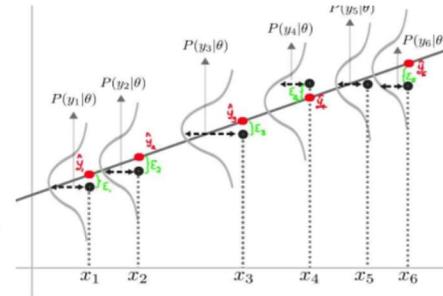
$$\hat{y} = \mathbf{w}^T \mathbf{x} + b + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

maximum likelihood

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

جای اینکه خودش را بیشینه کنیم منفی لگاریتمش را کمینه می‌کنیم

$$-\log P(\mathbf{y} | \mathbf{X}) = \sum_{i=1}^n \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)} - b)^2 \Rightarrow \min \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$



حال به محاسبه خواسته سوال میپردازیم.

منفی ضرر شرطی (log(-likelihood) برای رگرسیون تابع probit را می‌توان به صورت زیر بدست آورد:

با توجه به تعریف تابع probit :

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

که $N(\theta | 0,1)$ توزیع نرمال استاندارد است، احتمال یک نتیجه باینری y با توجه به بردار ورودی \vec{x} ، بردار ضرایب وزن ها \vec{w} و بایاس b را می‌توان به صورت زیر نشان داد:

$$P(y | \vec{x}, \vec{w}, b) = \begin{cases} \Phi(\vec{x} \cdot \vec{w} + b) & \text{if } y = 1 \\ 1 - \Phi(\vec{x} \cdot \vec{w} + b) & \text{if } y = 0 \end{cases}$$

تابع احتمال (likelihood) برای یک نقطه داده تنها (\vec{x}_i, y_i) به صورت زیر داده می‌شود:

$$\text{likelihood}(y_i | \vec{x}_i, \vec{w}, b) = \begin{cases} \Phi(\vec{x}_i \cdot \vec{w} + b) & \text{if } y_i = 1 \\ 1 - \Phi(\vec{x}_i \cdot \vec{w} + b) & \text{if } y_i = 0 \end{cases}$$

تابع فوق را می‌توانیم به شکل بازنویسی کنیم:

$$\text{likelihood}(y_i | \vec{x}_i, \vec{w}, b) = \Phi(\vec{x}_i \cdot \vec{w} + b)^{y_i} \cdot (1 - \Phi(\vec{x}_i \cdot \vec{w} + b))^{1-y_i}$$

با گرفتن لگاریتم منفی تابع احتمال (likelihood)، تابع منفی ضرر شرطی (log-likelihood) برای یک نمونه ($\mathcal{L}_i = \text{loss}((\vec{x}_i, y_i))$) به دست می‌آید: ($\mathcal{L} = \text{loss over all examples}$):

$$\mathcal{L}_i(\vec{w}, b) = -\log(\text{likelihood}(y_i | \vec{x}_i, \vec{w}, b))$$

$$\mathcal{L}_i(\vec{w}, b) = \begin{cases} -\log(\Phi(\vec{x}_i \cdot \vec{w} + b)) & \text{if } y_i = 1 \\ -\log(1 - \Phi(\vec{x}_i \cdot \vec{w} + b)) & \text{if } y_i = 0 \end{cases}$$

$$\mathcal{L}_i(\vec{w}, b) = -y_i \cdot \log(\Phi(\vec{x}_i \cdot \vec{w} + b)) - (1 - y_i) \cdot \log(1 - \Phi(\vec{x}_i \cdot \vec{w} + b))$$

حال برای محاسبه ضرر کل از میانگین ضرر ها (مجموع ضرر های هر یک از نمونه ها) روی تمام نمونه ها استفاده میکنیم.

$$\mathcal{L}(\vec{w}, b) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}_i(\vec{w}, b)$$

$$\mathcal{L}(\vec{w}, b) = -\frac{1}{n} \sum_{i=1}^n y_i \cdot \log(\Phi(\vec{x}_i \cdot \vec{w} + b)) + (1 - y_i) \cdot \log(1 - \Phi(\vec{x}_i \cdot \vec{w} + b))$$

که در آن n تعداد نمونه های داده است. این تابع ضرر log-likelihood برای رگرسیون پربویت را نشان می‌دهد.

** توجه کنید که Π در فرمول maximum likelihood با گرفتن لگاریتم به \sum تبدیل شده است.

به بیان دیگر و خلاصه تر:

For probit regression, the conditional distribution $p(y|x)$ is modeled as:

$$p(y=1|x) = \Phi(w^T x)$$

$$p(y=0|x) = 1 - \Phi(w^T x)$$

Where $\Phi(z)$ is the cumulative distribution function of the standard normal distribution.

The negative log-likelihood for a single observation (x, y) is:

$$-\log p(y|x)$$

Plugging in the probit regression probabilities, this becomes:

$$-y \log \Phi(w^T x) - (1-y) \log (1 - \Phi(w^T x))$$

So the negative conditional log-likelihood over the full dataset is:

$$-\sum_{i=1}^N [y_i \log \Phi(w^T x_i) + (1-y_i) \log (1 - \Phi(w^T x_i))]$$

and to calculate average loss over all examples we can divide it by n:

$$-\frac{1}{n} \sum_{i=1}^N [y_i \log \Phi(w^T x_i) + (1-y_i) \log (1 - \Phi(w^T x_i))]$$

منابع:

<https://chat.openai.com/>

<https://claude.ai/chat/>

سوال چهارم



$$\mathbf{X} \in \mathbb{R}^{n \times d}$$

پرسپترون چندلایه

$$\mathbf{H} = \mathbf{XW}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{H} \in \mathbb{R}^{n \times h}$$

$$\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$$

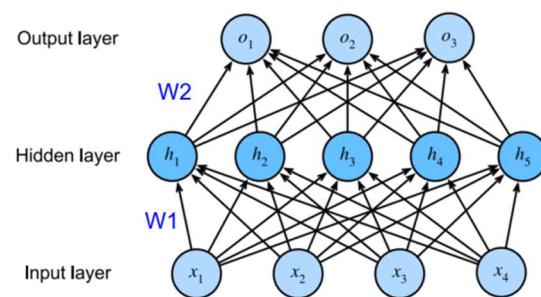
$$\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$$

$$\mathbf{O} = \mathbf{HW}^{(2)} + \mathbf{b}^{(2)}$$

$$\mathbf{O} \in \mathbb{R}^{n \times q}$$

$$\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$$

$$\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$$



$$\mathbf{O} = (\mathbf{XW}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

$$= \mathbf{XW}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

$$= \mathbf{XW} + \mathbf{b}$$

- دو لایه خطی متوالی هیچ تفاوتی با یک لایه ندارد

$$\text{linear(linear)} = \text{linear}$$



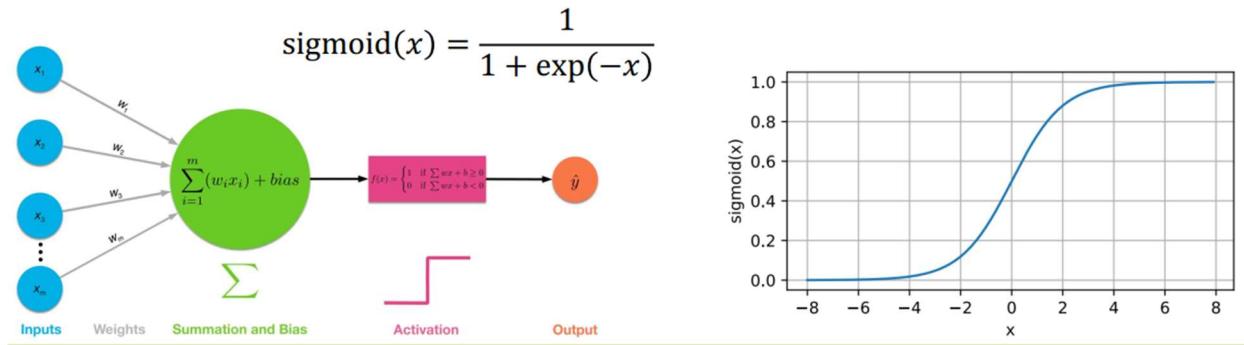
توابع فعال‌سازی

- توابع فعال‌سازی برای هر نورون مشخص می‌کنند که خروجی بخش خطی در چه شرایطی منجر به فعال شدن نورون شود
- به خصوص در شبکه‌های عمیق بسیار با اهمیت هستند

Sigmoid

- در اولین شبکه‌های عصبی، دانشمندان علاقه‌مند به مدل‌سازی نورون‌های بیولوژیکی بودند که یا فعال می‌شوند یا نمی‌شوند

- با توسعه روش‌های یادگیری مبتنی بر گرادیان، نیاز بود تا تقریب مشتق‌پذیر استفاده شود



(الف)

تابع فعال‌سازی جزء حیاتی شبکه‌های پرسپترون چند لایه (MLP) و دیگر معماری‌های شبکه عصبی هستند. آنها چندین هدف اساسی را انجام می‌دهند:

1. معرف غیر خطی بودن: بدون تابع فعال سازی، کل شبکه عصبی اساساً به یک مدل رگرسیون خطی کاهش می‌یابد، مهم نیست که چند لایه داشته باشد. توانایی مدل‌سازی روابط پیچیده و غیرخطی بین ورودی‌ها و خروجی‌ها یک نیاز اساسی برای بسیاری از مسائل دنیای واقعی است و تابع فعال‌سازی شبکه‌های عصبی را قادر می‌سازد تا چنین الگوهای پیچیده‌ای را ثبت کند.

2. فعال کردن شبکه برای یادگیری الگوهای پیچیده: با معرف غیرخطی‌ها، تابع فعال سازی به شبکه‌های عصبی اجازه می‌دهد تا توزیع‌های پیچیده و پیچیده‌تری را بیاموزند و نمایش دهند، که به ویژه در کارهای مانند تشخیص تصویر و گفتار، طبیعی اهمیت دارد. پردازش زبان و سایر وظایف پیچیده مدل سازی داده‌ها.

3. نرمال سازی: برخی از توابع فعال سازی به عادی سازی خروجی هر نورون کمک می‌کنند، که می‌تواند به سرعت بخشیدن به فرآیند یادگیری کمک کند. به عنوان مثال، تکنیک نرمال سازی دسته‌ای در ترکیب با تابع فعال سازی به تثبیت و تسریع آموخته شبکه‌های عصبی عمیق کمک می‌کند.

4. انتشار گرادیان: تابع فعال سازی نقش مهمی در انتشار کارآمد دارند که برای به روز رسانی وزن شبکه در طول فرآیند آموخته ضروری است. تابع فعال سازی که به درستی انتخاب شده‌اند می‌توانند به کاهش مسائلی مانند مشکل گرادیان ناپدید شونده، که در آن شبکه‌ها بسیار کوچک می‌شوند، کمک می‌کند و مانع از فرآیند آموخته در شبکه‌های عمیق می‌شود.

برخی از توابع فعال سازی متدائل مورد استفاده در شبکه‌های MLP عبارتند از تابع سینگوموئید، تابع تانژانت هیپربولیک (tanh)، واحد خطی اصلاح شده (ReLU)، Leaky ReLU وغیره. هر کدام از این توابع ویژگی‌های خاص خود را دارند و می‌توانند برای انواع مختلف وظایف و توزیع داده‌ها مناسب باشند. انتخاب تابع فعال سازی اغلب به ماهیت داده‌ها و نیازهای خاص مشکل در دست بستگی دارد.

در ادامه جواب این سوال به بیانی دیگر به زبان انگلیسی آمده است:

Activation functions are used in MLP networks for the following reasons:

- To introduce non-linearity: Without activation functions, MLP networks would simply be linear regression models, which are only able to learn linear relationships between the input and output data. Activation functions introduce non-linearity to the network, which allows it to learn complex patterns in the data.
- To increase the expressive power of the network: Activation functions allow MLP networks to represent a wider range of functions than linear regression models. This is because activation functions can introduce non-linearities, discontinuities, and other complex behaviors to the network.
- To improve the generalization performance of the network: Activation functions can help MLP networks to generalize better to unseen data. This is because activation functions can force the network to learn more abstract representations of the data.

Some common activation functions used in MLP networks include:

- Sigmoid function
- Tanh function
- ReLU function
- Leaky ReLU function
- ELU function

The choice of activation function depends on the specific task at hand and the desired properties of the network. For example, the ReLU function is often used in MLP networks for image recognition and natural language processing because it is computationally efficient and can help to prevent overfitting.

Here is an example to illustrate the importance of activation functions in MLP networks:

Imagine you are trying to train an MLP network to predict the price of a house based on its square footage, number of bedrooms, and number of bathrooms. Without activation functions, the network would only be able to learn a linear relationship between these features and the price of the house. This would be a very simplistic model, and it would not be able to accurately predict the price of houses with unusual features, such as houses with very large or small square footage.

However, if you add activation functions to the network, it will be able to learn more complex patterns in the data. For example, the network might learn that houses with more bathrooms tend to be more expensive than houses with fewer bathrooms, but that this effect is less pronounced for houses with very large square footage. The network might also learn that houses in certain neighborhoods tend to be more expensive than houses in other neighborhoods.

As a result of this increased expressive power, the network with activation functions will be able to predict the price of houses more accurately than the network without activation functions.

Overall, activation functions are essential for MLP networks to learn complex patterns in the data and to achieve high accuracy on prediction tasks.

(ب)

در تئوری، بله، هر تابع غیرخطی می‌تواند به عنوان یک تابع فعال سازی در یک شبکه عصبی استفاده شود. با این حال، انتخاب تابع فعال سازی باید چندین جنبه عملی و الزامات را در نظر بگیرد تا اطمینان حاصل شود که شبکه عصبی می‌تواند به طور موثر و کارآمد بگیرد.

در اینجا برخی از ملاحظات هنگام انتخاب یک تابع فعال سازی وجود دارد:

۱. تداوم و تفاوت پذیری(Continuity and differentiability): ترجیحاً تابع فعال سازی پیوسته و قابل تمايز باشد، زیرا این ویژگی استفاده از تکنیک‌های بهینه سازی مانند نزول گرادیان را در طول فرآیند آموزش تسهیل می‌کند.

۲. یکنواختی(Monotonicity): داشتن یک تابع فعال سازی یکنوا مفید است زیرا تضمین می‌کند که خروجی تابع فعال سازی در همان جهت ورودی تغییر می‌کند که می‌تواند فرآیند یادگیری را ساده کند.

۳. کارایی محاسباتی(Computational efficiency): پیچیدگی محاسباتی تابع فعال سازی می‌تواند به طور قابل توجهی برآموزش کلی و زمان استنتاج شبکه عصبی تأثیر بگذارد. توابع فعال سازی ساده و محاسباتی کارآمد اغلب برای کاربردهای عملی ترجیح داده می‌شوند.

۴. محدوده مقادیر خروجی(Range of output values): محدوده مقادیر خروجی از تابع فعال سازی باید برای کار خاص در دست مناسب باشد. به عنوان مثال، برای کارهای طبقه بندی بایزی، اغلب مفید است که یک تابع فعال سازی داشته باشیم که مقادیر را در محدوده بین ۰ و ۱ ترسیم کند.

در حالی که توابع فعال سازی کلاسیک مانند sigmoid، tanh، و ReLU به طور گسترده مورد استفاده قرار می‌گیرند، محققان همچنین توابع غیرخطی مختلف دیگری از جمله واحدهای خطی نمایی (ELUs)، softmax، swish، وغیره را بررسی کرده‌اند. این توابع جدیدتر برای رسیدگی به مسائل خاصی مانند مشکل گرادیان ناپدید شدن طراحی شده‌اند که آموزش کارآمدتر و عملکرد بهتر در شبکه‌های عصبی عمیق را ممکن می‌سازد.

در عمل، انتخاب تابع فعال سازی به الزامات خاص کار، ماهیت داده‌ها و جالش‌های بالقوه مرتبط با معماری شبکه بستگی دارد. اغلب شامل یک مبادله بین پیچیدگی محاسباتی، کارایی آموزش، و توانایی شبکه عصبی برای مدل‌سازی روابط غیرخطی پیچیده در داده‌ها است.

Theoretically, Yes, any non-linear function can be used as an activation function in a neural network. However, some non-linear functions are more suitable for use as activation functions than others. This is because some non-linear functions are more computationally efficient, have better generalization performance, and are less prone to overfitting.

Some common non-linear functions that are used as activation functions in neural networks include:

- Sigmoid function
- Tanh function
- ReLU function
- Leaky ReLU function
- ELU function
- Swish function
- Hardtanh function

- Softplus function

The choice of activation function depends on the specific task at hand and the desired properties of the network. For example, the ReLU function is often used in neural networks for image recognition and natural language processing because it is computationally efficient and can help to prevent overfitting.

Here are some factors to consider when choosing a non-linear function to use as an activation function:

- Computational efficiency: The activation function should be computationally efficient, especially if the network is large or needs to be trained on a large dataset.
- Generalization performance: The activation function should help the network to generalize better to unseen data.
- Risk of overfitting: The activation function should be less prone to overfitting, especially if the network is small or the dataset is noisy.
- Non-linearity: The activation function should be sufficiently non-linear to allow the network to learn complex patterns in the data.

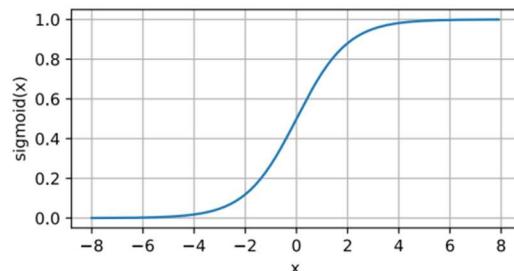
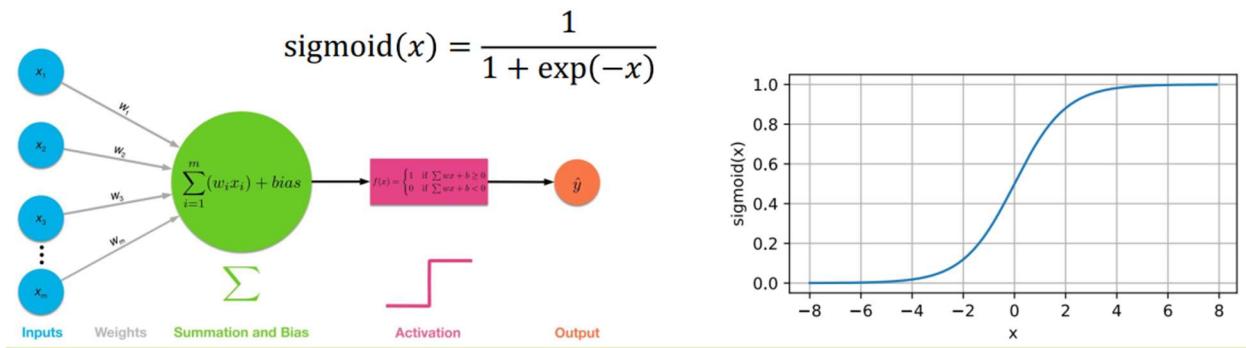
If you are unsure which non-linear function to use as an activation function, it is a good idea to experiment with different functions and see which one performs the best on your dataset. You can also use transfer learning to initialize the weights of the network with weights that have been pre-trained on a similar task. This will often give you a good starting point for choosing an activation function.

مراجع:

<https://chat.openai.com/>
<https://bard.google.com/>

Sigmoid

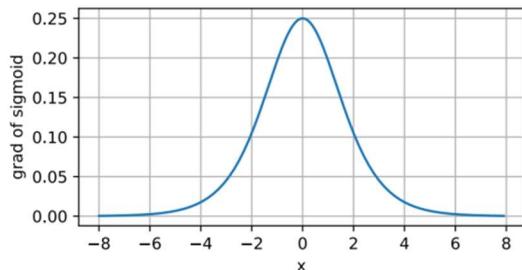
- در اولین شبکه‌های عصبی، دانشمندان علاقه‌مند به مدل‌سازی نورون‌های بیولوژیکی بودند که یا فعال می‌شوند یا نمی‌شوند
- با توسعه روش‌های یادگیری مبتنی بر گرادیان، نیاز بود تا تقریب مشتق‌پذیر استفاده شود



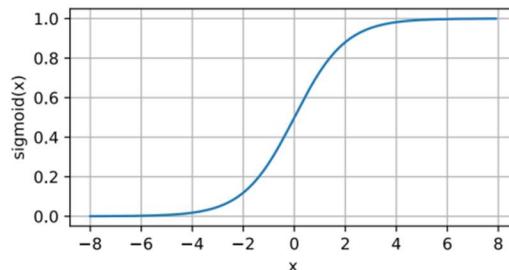
Sigmoid

- در بسیاری از مقادیر، گرادیان نزدیک به صفر است که بهینه‌سازی شبکه‌های عمیق را پیچیده می‌کند
- یکی از ایرادات sigmoid این است که خروجی آن همواره مثبت است

$$\frac{d}{dx} \text{sigmoid}(x) = \text{sigmoid}(x)(1 - \text{sigmoid}(x))$$

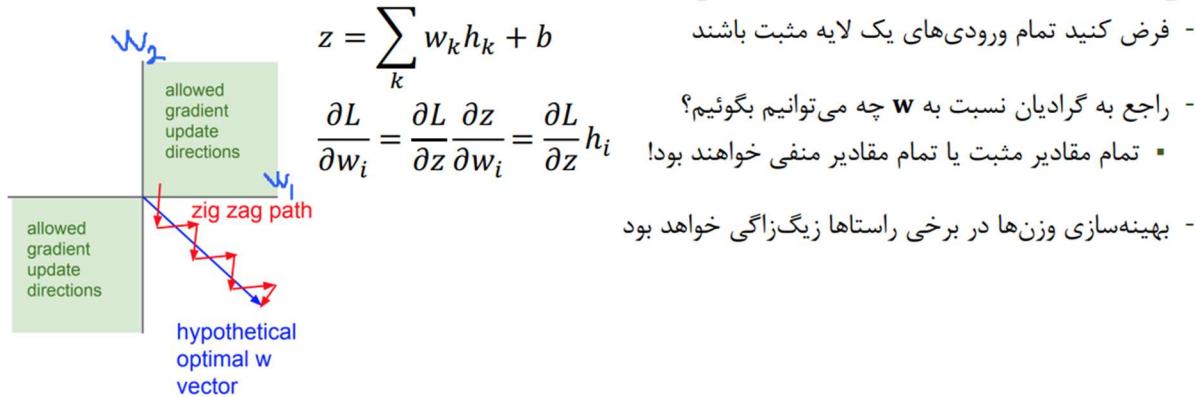


$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Sigmoid

- در بسیاری از مقادیر، گرادیان نزدیک به صفر است که بهینه‌سازی شبکه‌های عمیق را پیچیده می‌کند
- یکی از ایرادات sigmoid این است که خروجی آن همواره مثبت است

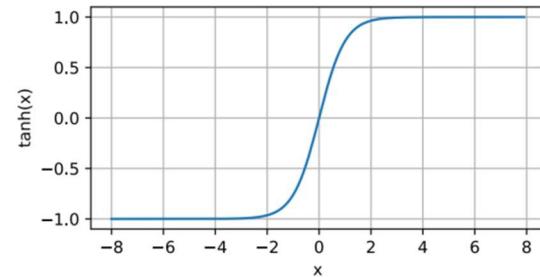
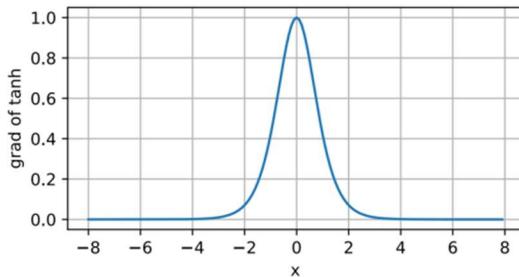


Tanh

- مشابه با تابع Sigmoid خروجی آن محدود است اما به بازه -1 تا $+1$
- اشباع شدن تابع \tanh ، بهینه‌سازی را دشوار می‌کند و از لحاظ محاسباتی هم به دلیل \exp پرهزینه است

$$\frac{d}{dx} \tanh(x) = 1 - \tanh^2(x)$$

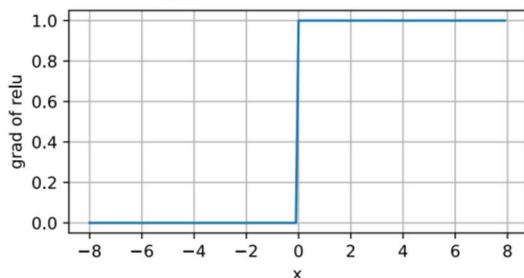
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} = 2\sigma(2x) - 1$$



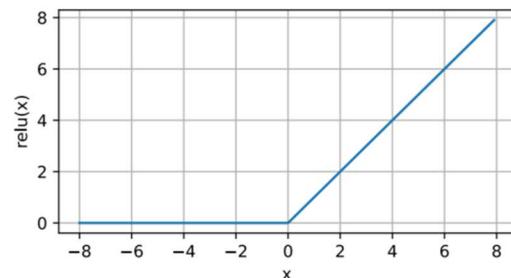
Rectified Linear Unit (ReLU)

- یک تابع غیرخطی بسیار ساده و پر کاربرد است
- بهینه سازی ReLU بسیار آسان است زیرا بسیار شبیه به واحد های خطی است
- مشتق ReLU برای مقادیری که فعال است همواره بزرگ است

$$\frac{d}{dx} \text{ReLU}(x) = x > 0$$



$$\text{ReLU}(x) = \max(x, 0)$$



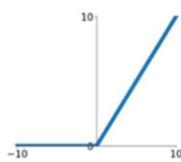
با شبیه مخالف صفر ReLU

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

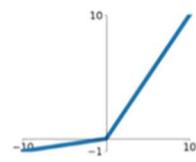
- تابع قدر مطلق ($g(z) = |z|$) با استفاده از $\alpha_i = -1$
- تابع از یک مقدار ثابت کوچک مانند $\alpha_i = 0.01$ با استفاده می کند
- نسخه Parametric ReLU (PReLU) را یک متغیر قابل آموزش در نظر می گیرد

$$h_i = g(\mathbf{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

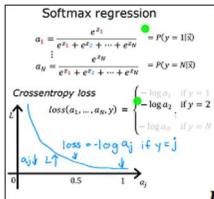
ReLU
 $\max(0, x)$



Leaky ReLU
 $\max(0.1x, x)$



دسته‌بندی چند کلاسه



- می‌خواهیم احتمال پسین مربوط به هر کلاس را تخمین می‌زنیم $P(y = i | \mathbf{x}) \in [0,1]$

برای آنکه مقادیر خروجی از جنس احتمال باشند (هر کدام نامنفی و مجموع برابر با ۱) می‌توانیم از تابع فعال‌سازی Sigmoid استفاده کنیم که تعمیم تابع Sigmoid است

$$\mathbf{o} = \mathbf{W}^T \mathbf{x} + \mathbf{b}$$

$$\hat{y}_i = \text{softmax}(\mathbf{o})_i = \frac{\exp(o_i)}{\sum_{k=1}^q \exp(o_k)}$$

- تابع ضرر متناسب با این تابع فعال‌سازی، categorical cross-entropy است
- $$l(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^q y_i \log \hat{y}_i$$

softmax مشتق

$$\begin{aligned} l(\mathbf{y}, \hat{\mathbf{y}}) &= - \sum_{i=1}^q y_i \log \frac{\exp(o_i)}{\sum_{k=1}^q \exp(o_k)} = \sum_{i=1}^q y_i \log \sum_{k=1}^q \exp(o_k) - \sum_{i=1}^q y_i o_i \\ &= \log \sum_{k=1}^q \exp(o_k) \sum_{i=1}^q y_i - \sum_{i=1}^q y_i o_i = \log \sum_{k=1}^q \exp(o_k) - \sum_{i=1}^q y_i o_i \end{aligned}$$

$$\partial_{o_i} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_i)}{\sum_{k=1}^q \exp(o_k)} - y_i = \text{softmax}(\mathbf{o})_i - y_i$$

- مشابه با رگرسیون خطی، گرادیان تابع ضرر نسبت به خروجی بخش خطی برابر با میزان اختلاف پیش‌بینی با مقدار واقعی است

اجازه دهید قبل از مقایسه، هر یک از این تابع‌های فعال‌سازی به همراه مزايا و معایب آنها، توضیح دهیم:

۱. تابع سیگموئید:

- توضیح: تابع سیگموئید، که به صورت $\text{sigmoid}(x) = 1 / (1 + \exp(-x))$ نشان داده می‌شود، ورودی را در محدوده‌ای بین ۰ و ۱ نگاشت می‌کند. از لحاظ تاریخی به طور گستردگی به عنوان یک تابع فعال‌سازی استفاده می‌شود.

- مزايا: یک گرادیان صاف(smooth) را ارائه می‌دهد که می‌تواند برای الگوريتم‌های بهینه سازی ساده مبتنی بر گرادیان مفید باشد. همچنانی یک خروجی بازیزی واضح را ارائه می‌دهد که می‌تواند به عنوان احتمال تفسیر شود.

- معایب: خروجی‌های سیگموئید صفر محور نیستند که می‌تواند منجر به بروز مشکلاتی در به روز رسانی وزن در حین بهینه سازی شود. مستعد مشکل گرادیان ناپدید شدن است، به ویژه در شبکه‌های عمیق، که مانع یادگیری می‌شود.

۲. تابع Softmax:

- توضیح: تابع Softmax $\text{Softmax}(x) = \exp(x) / \sum(\exp(x))$ در مسائل طبقه بندی چند کلاسه برای تبدیل نمرات خام به احتمالات استفاده می شود. خروجی ها را در یک توزیع احتمال عادی می کند.

- مزایا: یک توزیع احتمال نرمال شده را ارائه می دهد و آن را برای مسائل طبقه بندی چند طبقه ایده آل می کند. زمانی که خروجی باید توزیع احتمال را در چندین کلاس نشان دهد، یک انتخاب طبیعی است.

- معایب: به نقاط دورافتاده حساس است و می تواند شبیه های کوچکی را در طول آموزش ایجاد کند، که آن را مستعد مشکل گرادیان ناپدید، به ویژه در شبکه های عمیق تر می کند.

۳. تابع ReLU (واحد خطی اصلاح شده):

- توضیح: فعال سازی ReLU به صورت $\text{ReLU}(x) = \max(0, x)$ تعریف می شود، به این معنی که x را برای ورودی های منفی و خود ورودی را برابر هر ورودی مثبت بر می گرداند.

- مزایا: از نظر محاسباتی کارآمد است و امکان آموزش سریع تر شبکه های عصبی عمیق را فراهم می کند. این مشکل شبیه ناپدید شدن را کاهش می دهد و آن را برای آموزش شبکه های عصبی مناسب می کند. همچنین فعال سازی های پراکنده ای را ارائه می دهد که می تواند در سناریوهای خاص مفید باشد.

- معایب: واحدهای ReLU می توانند از مشکل "dying ReLU" رنج ببرند، که در آن تعداد زیادی واحد می توانند برای هر ورودی خروجی x داشته باشند و به طور موثر آنها را غیرفعال می کند و منجر به سلول های عصبی مرده می شود. این می تواند مشکلاتی را با ظرفیت نمایندگی مدل ایجاد کند.

۴. تابع Tanh:

- توضیح: تابع Tanh شبیه تابع سیگموئید است اما ورودی را در محدوده $-1 < x < 1$ می کند که به صورت $\tanh(x) = (\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x))$

- مزایا: صفر محور است که مدلسازی ورودی هایی که دارای مقادیر شدیداً منفی، خنثی و به شدت مثبت هستند را آسان تر می کند. همچنین در مقایسه با تابع سیگموئید کمتر مستعد مشکل گرادیان ناپدید شدن است.

- معایب: مشابه تابع sigmoid، می تواند برای ورودی های شدید اشباع شود، که منجر به ناپدید شدن مشکل گرادیان در شبکه های عمیق می شود. از نظر محاسباتی به اندازه ReLU کارآمد نیست.

مقایسه:

- توابع سیگموئید و tanh از مشکل گرادیان ناپدید شدن رنج می برد، به خصوص در شبکه های عمیق تر.

- از نظر محاسباتی کارآمد است و مشکل گرادیان ناپدید شدن را کاهش می دهد، اما می توانند از مشکل "dying ReLU" رنج ببرد.

- Softmax برای کارهای طبقه بندی چند کلاسه مناسب است، اما می تواند به موارد پرت حساس باشد و از شبیه ناپدید شدن رنج می برد.

انتخاب یک تابع فعال سازی به نیازهای خاص مشکل، ویژگی های داده ها و چالش های بالقوه مرتبط با معماری شبکه بستگی دارد.

در ادامه جواب این سوال به بیانی دیگر به زبان انگلیسی آمده است:

Sigmoid Function

The sigmoid function is a non-linear function that squashes its input to a value between 0 and 1. It is defined as follows:

$$\text{sigmoid}(x) = 1 / (1 + \exp(-x))$$

Advantages of the sigmoid function:

- It is a simple and well-understood function.
- It is computationally efficient.
- It outputs values between 0 and 1, which can be useful for tasks such as binary classification.

Disadvantages of the sigmoid function:

- It can saturate for large inputs, which can lead to vanishing gradients during training.
- It is not zero-centered, which can make it difficult to train deep neural networks.

Softmax Function

The softmax function is a non-linear function that converts a vector of inputs to a vector of probabilities. It is defined as follows:

$$\text{softmax}(x) = \exp(x) / \sum(\exp(x))$$

Advantages of the softmax function:

- It outputs a vector of probabilities, which can be useful for tasks such as multi-class classification.
- It is numerically stable, meaning that it does not produce large or small values that can lead to numerical errors.

Disadvantages of the softmax function:

- It is computationally more expensive than the sigmoid function.
- It can be sensitive to the scale of the inputs.

ReLU Function

The ReLU function is a non-linear function that outputs the input if it is positive, and 0 otherwise. It is defined as follows:

$$\text{ReLU}(x) = \max(0, x)$$

Advantages of the ReLU function:

- It is computationally very efficient.
- It is less prone to saturation than the sigmoid function, which can help to prevent vanishing gradients during training.
- It is zero-centered, which can make it easier to train deep neural networks.

Disadvantages of the ReLU function:

- It can die if the input is always negative.
- It can be noisy for small inputs.

Tanh Function

The tanh function is a non-linear function that squashes its input to a value between -1 and 1. It is defined as follows:

$$\tanh(x) = (\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x))$$

Advantages of the tanh function:

- It outputs values between -1 and 1, which can be useful for tasks such as regression.
- It is less prone to saturation than the sigmoid function, which can help to prevent vanishing gradients during training.
- It is zero-centered, which can make it easier to train deep neural networks.

Disadvantages of the tanh function:

- It is computationally more expensive than the sigmoid function.
- it can saturate for extreme inputs, leading to the vanishing gradient problem in deep networks.

Comparison

The following table compares the four activation functions:

Activation function	Output range	Advantages	Disadvantages
Sigmoid	0 to 1	Simple, computationally efficient, outputs probabilities	Saturates for large inputs, not zero-centered
Softmax	0 to 1, probabilities	Numerically stable, outputs probabilities	Computationally expensive, sensitive to the scale of the inputs
ReLU	0 and above	Computationally efficient, less prone to saturation, zero-centered	Can die if the input is always negative, noisy for small inputs
Tanh	-1 to 1	Outputs values between -1 and 1, less prone to saturation than sigmoid, zero-centered	Computationally more expensive than the sigmoid function

Which activation function to use?

The best activation function to use depends on the specific task at hand and the desired properties of the network. For example, the ReLU function is often used for image recognition and natural language processing because it is computationally efficient and can help to prevent overfitting. The softmax function is often used for multi-class classification because it outputs probabilities.

If you are unsure which activation function to use, it is a good idea to experiment with different functions and see which one performs the best on your dataset. You can also use transfer learning to initialize the weights of the network with weights that have been pre-trained on a similar task. This will often give you a good starting point for choosing an activation function.

(ب)

برای پیاده سازی توابع از تعاریف داخل اسلاید های درس و همچنین جواب های قسمت الف همین سوال استفاده کردیم. پیاده سازی هر تابع در حد چندین خط کد ساده بود شمال عملگر های /, -, *, + به همراه استفاده از توابع `max()`, `sum()`, and `exp()` بود. هم میتوان از توابع `numpy` و هم از توابع `torch` استفاده کرد. فقط در استفاده از توابع به یک ارور خوردم که باید بجای ورودی به عنوان عدد از `tensor` استفاده میکردم. ارور به شرح زیر بود:

```
TypeError: max() received an invalid combination of arguments - got (int,
Tensor), but expected one of: * (Tensor input, *, Tensor out) * (Tensor
input, Tensor other, *, Tensor out) * (Tensor input, int dim, bool
keepdim, *, tuple of Tensors out) * (Tensor input, name dim, bool keepdim,
*, tuple of Tensors out)
```

توضیحات بیشتر:
تابع سیگموئید:

```
def sigmoid(x):
    return 1 / (1 + torch.exp(-x))
```

تابع سیگموئید یک تابع ریاضی است که هر عدد با ارزش واقعی را به مقداری بین ۰ و ۱ نگاشت می‌کند. معمولاً به عنوان یک تابع فعال سازی در شبکه‌های عصبی استفاده می‌شود.

تابع یک ورودی x می‌گیرد.

خط $(\text{torch.exp}(-x))$ نمایی $-x$ را محاسبه می‌کند.

خط $(1 / (\text{torch.exp}(-x) + 1))$ تابع سیگموئید را با تقسیم ۱ بر مجموع ۱ و نمایی $-x$ محاسبه می‌کند.

نتیجه خروجی تابع سیگموئید است.

عملکرد Softmax :

```
def softmax(x):
    return torch.exp(x) / torch.sum(torch.exp(x))
```

تابع softmax اغلب برای تبدیل بردار اعداد واقعی به توزیع احتمال استفاده می‌شود.

تابع یک ورودی x می‌گیرد که یک تانسور یا بردار است.

خط $(\text{torch.exp}(x))$ هر عنصر را در بردار ورودی x محاسبه می‌کند.

خط $(\text{torch.sum}(\text{torch.exp}(x)))$ مجموع مقادیر توان را در بردار محاسبه می‌کند.

در نهایت، $(\text{torch.sum}(\text{torch.exp}(x)) / \text{torch.sum}(\text{torch.exp}(x)))$ هر عنصر در بردار دارای توان را بر مجموع تمام مقادیر توان یافته تقسیم می‌کند و توزیع احتمالی را تولید می‌کند که در آن انتگرال خروجی ۱ است.

تابع ReLU (واحد خطی اصلاح شده):

```
def ReLU(x):
    return torch.max(torch.tensor(0), x)
```

تابع ReLU یک تابع فعال سازی غیر خطی است که معمولاً در شبکه‌های عصبی استفاده می‌شود.

تابع یک ورودی x می‌گیرد.

خط $(\text{torch.max}(\text{torch.tensor}(0), x))$ هر عنصر x را با ۰ مقایسه می‌کند و حداقل دو مقدار را برمی‌گرداند.

اگر عنصری در x کمتر از ۰ باشد، با ۰ جایگزین می‌شود. در غیر این صورت، عنصر بدون تغییر باقی می‌ماند.

تانسور حاصل خروجی تابع ReLU است.

تابع Leaky ReLU :

```
def LeakyReLU(x):
    return torch.max(0.01 * x, x)
```

تابع Leaky ReLU گونه‌ای از تابع ReLU است که به مقادیر منفی کوچک اجازه عبور می‌دهد و از مشکل "ReLU در حال مرگ" جلوگیری می‌کند.

تابع یک ورودی x می‌گیرد.

خط $0.01 * x$ هر عنصر x را در 0.01 ضرب می‌کند.

خط $(x * \text{torch.max}(0.01 * x, 0))$ هر عنصر x را با عنصر مربوطه در x مقایسه می‌کند و حداکثر دو مقدار را برمی‌گرداند. اگر عنصری در x کمتر از 0 باشد، 0 برابر عنصر جایگزین می‌شود. در غیر این صورت، عنصر بدون تغییر باقی می‌ماند. تansor حاصل خروجی تابع Leaky ReLU است.

تابع تانژانت هایپربولیک (\tanh):

```
def tanh(x):
    return (1 - torch.exp(-2*x)) / (1 + torch.exp(-2*x))
```

تابع مماس هذلولی (تانژانت هایپربولیک)، \tanh ، یک تابع فعال سازی غیر خطی شبیه تابع سیگموئید است. تابع یک ورودی x می‌گیرد.

خط $\text{torch.exp}(-2*x)$ نمایی $-2*x$ را محاسبه می‌کند.

خط $(1 - \text{torch.exp}(-2*x))$ تفاوت بین 1 و نمایی 2^x را محاسبه می‌کند.

خط $((1 - \text{torch.exp}(-2*x)) / (1 + \text{torch.exp}(-2*x)))$ اختلاف فوق را بر مجموع 1 و نمایی 2^x - تقسیم می‌کند.

تansor حاصل خروجی تابع \tanh است که مقادیر را به محدوده ای بین -1 و 1 نگاشت می‌کند.

(ج)

معماری برای یک شبکه MLP ساده برای جداسازی کاراکترهای (حروف) ب، گ و ص:

تعداد لایه‌ها: ۳

دلیل: این یک مجموعه داده نسبتاً کوچک با تنها ۳ کلاس است، بنابراین یک شبکه ۳ لایه باید کافی باشد. لذا از یک لایه ورودی (input layer)، یک لایه پنهان (hidden layer) و یک لایه خروجی (output layer) استفاده می‌کنیم. افزودن لایه‌های بیشتر می‌تواند منجر به پیچیدگی محاسباتی، پیچیدگی بیش از حد مدل و بیش از حد برازش (overfitting) شود.

تعداد نورون‌ها در هر لایه:

لایه ورودی: ۱۰۲۴ نورون یعنی یک نورون برای هر پیکسل در تصویر (چون تصاویر ورودی را graysacle (تک کanalه) و به سایز ۳۲ در ۳۲ resize کردیم) لایه پنهان: ۲۵۶ نورون

لایه خروجی: ۳ نورون (یک نورون برای هر کلاس (ب، گ و ص))

دلیل: لایه ورودی باید نورون‌های کافی برای نمایش تصویر ورودی (به تعداد پیکسل‌ها) داشته باشد. لایه پنهان می‌تواند تعداد نورون‌های کمتری نسبت به لایه ورودی داشته باشد، اما همچنان باید نورون‌های کافی برای یادگیری الگوهای پیچیده در داده‌ها داشته باشد. لایه خروجی باید برای هر کلاس یک نورون داشته باشد تا احتمال هر کلاس را پیش بینی کند.

تابع فعال سازی: ReLU

دلیل: تابع فعال سازی ReLU از نظر محاسباتی کارآمد است و نسبت به تابع فعال سازی سیگموئید کمتر مستعد ناپدید شدن گردیدن‌ها است. و به طور معمول انتخاب مناسبی برای لایه‌های میانی شبکه‌های عصبی است. (می‌توانستیم برای لایه خروجی از softmax نیز استفاده کنیم تا اعداد خروجی (logits) تبدیل به توزیع احتمالاتی شوند ولی اجباری نیست و فقط هزینه محاسباتی دارد چراکه با گرفتن ماکریم logit‌ها می‌توانیم کلاس پیش بینی شده با بیشترین احتمال را پیدا کنیم به عبارتی بزرگترین logit بعد از عبور از تابع softmax باز هم بزرگترین مقدار را دارد چون تابع softmax اکیداً صعودی است.)

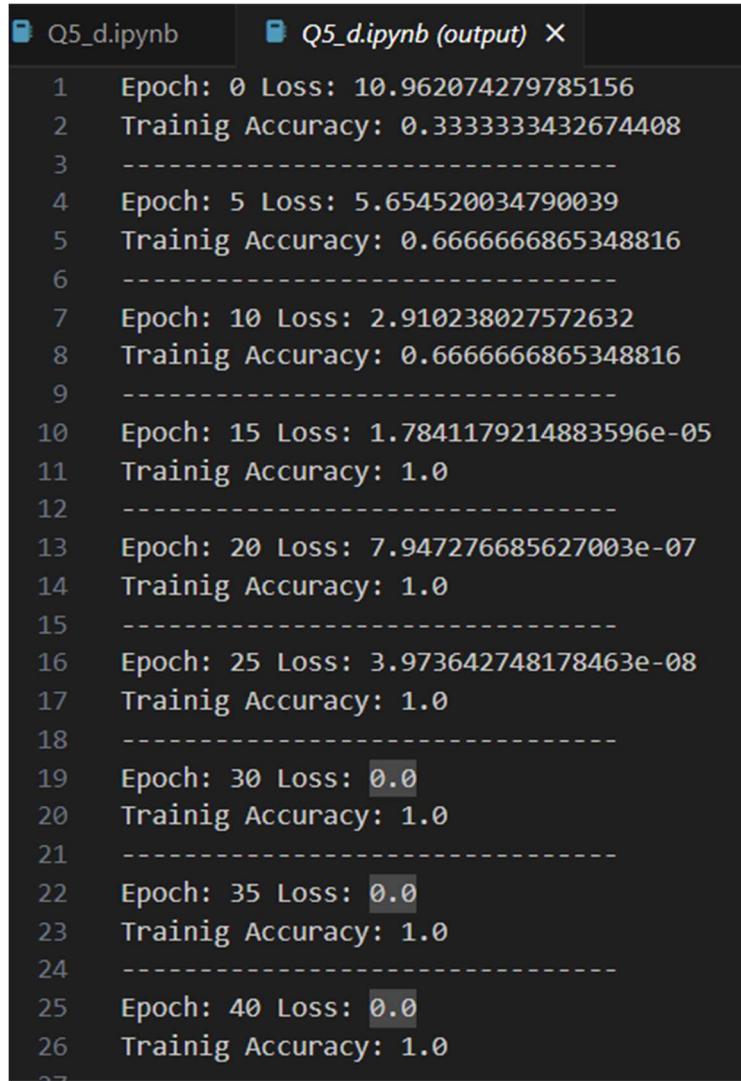
تابع ضرر: Categorical Cross-entropy loss

دلیل: تابع Categorical Cross-entropy loss معمولاً برای کارهای طبقه بندی چند کلاسه (multi-class classification) استفاده می شود. این یک انتخاب خوب برای این کار است زیرا اندازه گیری می کند که چگونه پیش بینی های مدل با برچسب های واقعی مطابقت دارند.

نتیجه:

این شبکه ساده MLP قادر است به طور موثر کارکترهای (حروف) ب، گ و ص را از هم جدا کند. تابع فعال سازی ReLU و تابع از Categorical Cross-entropy loss برای این کار مناسب هستند.

(۵)



```
Q5_d.ipynb Q5_d.ipynb (output) ×
1 Epoch: 0 Loss: 10.962074279785156
2 Trainig Accuracy: 0.3333333432674408
3 -----
4 Epoch: 5 Loss: 5.654520034790039
5 Trainig Accuracy: 0.6666666865348816
6 -----
7 Epoch: 10 Loss: 2.910238027572632
8 Trainig Accuracy: 0.6666666865348816
9 -----
10 Epoch: 15 Loss: 1.7841179214883596e-05
11 Trainig Accuracy: 1.0
12 -----
13 Epoch: 20 Loss: 7.947276685627003e-07
14 Trainig Accuracy: 1.0
15 -----
16 Epoch: 25 Loss: 3.973642748178463e-08
17 Trainig Accuracy: 1.0
18 -----
19 Epoch: 30 Loss: 0.0
20 Trainig Accuracy: 1.0
21 -----
22 Epoch: 35 Loss: 0.0
23 Trainig Accuracy: 1.0
24 -----
25 Epoch: 40 Loss: 0.0
26 Trainig Accuracy: 1.0
27
```

با استفاده از کتابخانه PyTorch مدل توضیح داده شده را پیاده سازی کردیم. همان طور که مشاهده میکنید loss ها با گذشت Epoch ها کاهش می باید و دقت (Training accuracy) هم در حال افزایش هست به گونه ای که از Epoch 30 به بعد دقت (Training accuracy) روی تصاویر داده شده به ۱۰۰ درصد رسیده است و Cross Entropy Loss هم به ۰ رسیده است. کد پیاده سازی شده در فایل Q5_d.ipynb ضمیمه شده موجود است و به طور کامل کامنت گذاری و توضیح داده شده است. با این حال در زیر به طور خلاصه آن را توضیح می دهیم.

توضیح کد:

چهار خط اول کتابخانه های لازم را وارد می کنند:

کتابخانه torch: torch

کتابخانه شبکه عصبی torch.nn: PyTorch

PIL: کتابخانه تصویربرداری پایتون برای دستکاری تصویر

numpy: کتابخانه NumPy برای محاسبات علمی

تابع flatten_image() یک تصویر PIL را به یک آرایه ۱ بعدی NumPy تبدیل می کند. این امر ضروری است زیرا شبکه های عصبی PyTorch فقط تansورها را به عنوان ورودی می پذیرند. این تابع ابتدا تصویر را به مقیاس خاکستری تبدیل می کند و سپس آن را به یک آرایه NumPy تبدیل می کند. در نهایت، آرایه را به یک بردار ۱ بعدی مسطح می کند.

کلاس MLP معماری شبکه MLP را تعریف می کند. شبکه دارای سه لایه است: یک لایه ورودی، یک لایه پنهان و یک لایه خروجی. لایه ورودی دارای ۱۰۲۴ نورون است که با تعداد پیکسل های یک تصویر 32×32 در مقیاس خاکستری مطابقت دارد. لایه پنهان دارای ۲۵۶ نورون و لایه خروجی دارای ۳ نورون است که با تعداد کلاس های مسئله طبقه بندی چند طبقه مطابقت دارد.

متده forward() کلاس MLP مسیر عبور از شبکه را تعریف می کند. پس جلو به سادگی داده های ورودی را می گیرد و از لایه های مختلف شبکه عبور می دهد و تابع فعال سازی را بعد از هر لایه اعمال می کند. خروجی پاس رو به جلو کلاس پیش بینی شده برای هر تصویر ورودی است.

متغیر device مشخص می کند که از کدام دستگاه برای آموزش و ارزیابی شبکه استفاده شود. اگر یک GPU در دسترس باشد، شبکه بر روی GPU آموزش داده می شود. در غیر این صورت، شبکه روی CPU آموزش داده می شود.

لیست images حاوی تصاویر آموزشی است. تصاویر آموزشی با استفاده از تابع Image.open() از سیستم فایل بارگذاری می شوند. سپس تصاویر به 32×32 پیکسل تغییر اندازه داده می شوند و با استفاده از تابع flatten_image() به بردارهای ۱ بعدی مسطح می شوند.

تansور images با استفاده از تابع torch.tensor() به تansور PyTorch تبدیل می شود. تansور نیز به device مشخص شده توسط متغیر دستگاه منتقل می شود.

تansور labels حاوی برچسب های آموزشی است. برچسب های آموزشی از یک آرایه NumPy بارگیری می شوند و با استفاده از تابع torch.tensor() به یک تansور PyTorch تبدیل می شوند. تansور نیز به دستگاه مشخص شده توسط متغیر device منتقل می شود.

متغیر model نمونه ای از کلاس MLP است. مدل نیز به دستگاه مشخص شده توسط متغیر device منتقل می شود.

متغیر criterion تابع ضرر را برای آموزش شبکه مشخص می کند. تابع تلفات متقابل آنتروپی معمولاً برای مسائل طبقه بندی چند طبقه استفاده می شود.

متغیر optimizer بینه ساز را برای آموزش شبکه مشخص می کند. بینه ساز Adam یک بینه ساز محبوب است که اغلب برای آموزش شبکه های عصبی استفاده می شود.

متدها (model.train) مدل را روی حالت آموزش قرار می دهد. این مهم است زیرا برخی از لایه ها، مانند لایه های dropout، در طول آموزش و ارزیابی رفتار متفاوتی دارند.

حلقه آموزش روی داده های آموزشی برای ۱۰۰ دوره تکرار می شود. در هر دوره مراحل زیر انجام می شود:

۱. این مدل با دسته ای از تصاویر آموزشی تغذیه می شود.
۲. مدل پیش بینی های خود را برای تصاویر آموزشی خروجی می دهد.
۳. اتلاف با استفاده از تابع تلفات متقابل آنتروپی محاسبه می شود.
۴. شبکه تلفات با توجه به پارامترهای مدل محاسبه می شود.
۵. پارامترهای مدل با استفاده از بینه ساز Adam به روز می شوند.

بعد از هر ۵ دوره، training loss and training accuracy روی کنسول چاپ می شود.

حلقه ارزیابی مدل را روی تصاویر آموزشی ارزیابی می کند. مدل ابتدا با استفاده از روش eval (model.eval) روی حالت ارزیابی تنظیم می شود. این مهم است زیرا برخی از لایه ها، مانند لایه های dropout، در طول آموزش و ارزیابی رفتار متفاوتی دارند.

سپس مدل با تصاویر آموزشی تغذیه می شود و برجسب های پیش بینی شده محاسبه می شوند. سپس دقت مدل بر روی تصاویر آموزشی محاسبه شده و در کنسول چاپ می شود.

منابع:

<https://bard.google.com>
<https://chat.openai.com>
<https://claude.ai/chats>

- $z \in (-\infty, +\infty) \Rightarrow \text{RELU}(z) \in [0, +\infty)$
- $\Rightarrow \sigma(\text{RELU}(z)) \in [0.5, 1] \xrightarrow{\text{threshold}=0.5} \sigma(\text{RELU}(z)) \geq 0.5 \quad \hat{y} = 1$
predicted class is always 1.
- هیچ وقت خروجی MLP کلاس 0 را پیش بینی نمی کنی.
- $\text{RELU}(z)$
- $\sigma(\text{RELU}(z)) = y$
- $\text{RELU}(x) = \max(0, x)$
- $\sigma(x) = \frac{1}{1 + e^{-x}} \quad 0.5 \leq y < 1$
- همان طور که در مودارهای بالا می بینیم
- $y = \sigma(\text{RELU}(z)) \geq 0.5 \xrightarrow{\text{threshold}=0.5} \text{predicted class} = 1$
- class = $\begin{cases} 1 & y \geq 0.5 \\ 0 & y < 0.5 \end{cases}$
- مشکل: این مدل MLP همواره به ازای هر ورودی کلاس 1 را به عنوان خروجی پیش بینی می کند و هیچ وقت کلاس 0 را پیش بینی نمی کند.
- مشکل: این مشکل با آموزش دیگر هم اصلاح نمی شود مگر آنکه آستانه را عوض کنیم مثلاً 0.75 بدگذاریم. بنابراین این نوع تولیپ توابع فعال سازی کار معقولی نیست و نتیجه خوبی نمی دهد. مگر در حالت خیلی خاص مثلاً PAPCO
- 99% داده ها مربوط به کلاس 1 باشند.

ترکیب تابع فعال‌سازی ReLU و تابع فعال‌سازی سیگموئید، همراه با آستانه ۵٪. برای طبقه‌بندی باینری، می‌تواند چالش‌ها و مسائلی را در فرآیند طبقه‌بندی ایجاد کند:

۱. مشکل گرادیان ناپدید شدن(Vanishing Gradient Problem): تابع فعال‌سازی ReLU برای ایجاد مشکل گرادیان ناپدید شدن، به ویژه برای ورودی‌های منفی شناخته شده است. این می‌تواند منجر به همگرایی کند یا ناکارآمد در طول آموزش، به ویژه در شبکه‌های عمیق‌تر شود. ترکیب ReLU با تابع sigmoid ممکن است این مشکل را تقویت کند و به طور بالقوه آموزش موثر شبکه را دشوارتر کند.

۲. ترکیب Non-Smooth Transitions: ترکیب ReLU و sigmoid می‌تواند انتقال‌های غیر هموار در اطراف آستانه ۵٪ ایجاد کند. این می‌تواند منجر به تغییرات ناگهانی در خروجی شود و باعث بی‌ثباتی شود و به طور بالقوه بر همگرایی شبکه در طول فرآیند آموزش تأثیر بگذارد. ممکن است منجر به پیش‌بینی‌های کمتر دقیق و ناپایدارتر شود.

۳. از دست دادن اطلاعات(Loss of Information): استفاده از یک آستانه سخت ۵٪ برای طبقه‌بندی ممکن است منجر به از دست رفتن اطلاعات شود، به خصوص در سناریوهایی که کلاس‌ها کاملاً قابل تفکیک نیستند. این می‌تواند منجر به طبقه‌بندی اشتباہ برخی از نقاط داده ای شود که به مرز تصمیم نزدیک می‌شوند و منجر به کاهش دقت طبقه‌بندی می‌شود.

۴. پیچیدگی در بازنمایی‌های یادگیری(Complexity in Learning Representations): ترکیب ReLU و sigmoid می‌تواند یادگیری موثر بازنمایی‌های پیچیده را برای شبکه چالش برانگیز کند. این می‌تواند قدرت بیان شبکه عصبی را محدود کند و مانع از توانایی آن در گرفتن الگوهای روابط پیچیده در داده‌ها شود.

۵. اشباع سیگموئید(Sigmoid saturation): سیگموئید در ۰ و ۱ اشباع می‌شود. این بدان معنی است که هر ورودی منفی یا مثبت بزرگ به سیگموئید به ۰ یا ۱ فشرده می‌شود. اگر سیگموئید در این رژیم اشباع(saturated regime) کار کند، توانایی تمایز را از دست می‌دهد و می‌تواند یادگیری را مختل کند.

۶. کالیبراسیون اطمینان(Confidence calibration): MLP ممکن است بیش از حد مطمئن شود، با خروجی‌های سیگموئید نزدیک به ۰ یا ۱ حقیقت در صورت نادرست. آستانه ۵٪. این را تشدید می‌کند - خروجی ۵۱٪ اطمینان بالای دارد، حتی اگر احتمال واقعی کمی بالاتر از ۵٪ باشد.

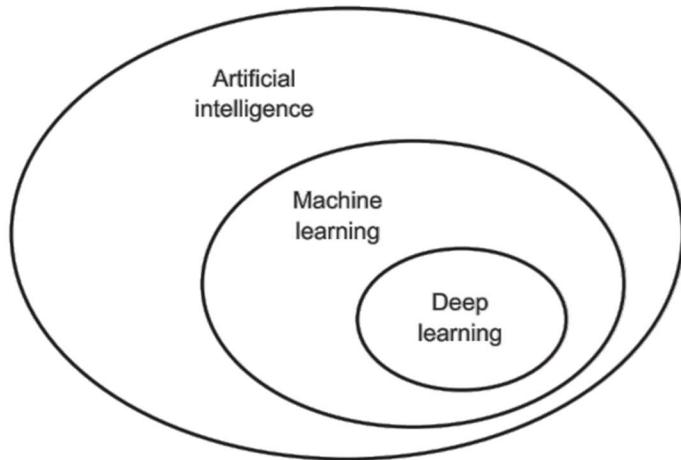
۷. مرکزیت آستانه(Threshold centering): با آستانه ۵٪، MLP تمایل دارد به جای توزیع احتمال واقعی، خروجی‌ها را حدود ۵٪ متمرکز کند. این سوگیری می‌تواند توانایی تبعیض را کاهش دهد.

برای پرداختن به این چالش‌ها، بررسی توابع فعال‌سازی جایگزین که می‌توانند مثل ELU مشکل گرادیان ناپدید شونده را کاهش دهند، انتقال نرم‌تر را فراهم کنند و مرز تصمیم‌گیری انعطاف‌پذیری را فراهم کنند، ممکن است مفید باشد. علاوه بر این، در نظر گرفتن سایر مقادیر آستانه یا ترکیب تکنیک‌های مانند تنظیم آستانه تصمیم به صورت پویا یا استفاده از استراتژی‌های طبقه‌بندی پیشرفته تر می‌تواند به بهبود عملکرد و پایداری شبکه در کار طبقه‌بندی کمک کند.

منابع:

<https://chat.openai.com/>

هوش مصنوعی، یادگیری ماشین، یادگیری عمیق



یادگیری عمیق

- زیرشاخه‌ای از یادگیری ماشین است که مبتنی بر یادگیری لایه‌های متوالی از بازنمایی‌های معنادار است
- در بسیاری از مسائل یادگیری ماشین توانسته است نتایج لبه دانش را بدست بیاورد
- یادگیری عمیق لزوماً به معنای درک عمیق‌تری نیست!
- ایده یادگیری سلسله‌مراتبی مفاهیم به رایانه اجازه می‌دهد تا مفاهیم پیچیده را از مفاهیم ساده‌تر بسازد



بازنمایی چیست؟

- تمام الگوریتم‌های یادگیری مارشین شامل یافتن چنین تبدیل‌هایی است به نحوی که داده‌ها به یک بازنمایی بهتر برای مسئله مورد نظر تبدیل شوند

- تغییر مختصات، نگاشت‌های خطی (ممکن است برخی از اطلاعات حذف شود)، عملگرهای غیرخطی، و ...

- الگوریتم‌های ML معمولاً برای یافتن این تبدیل‌ها خلاق نیستند

- فقط در یک مجموعه از تبدیل‌های از پیش تعریف شده جستجو می‌کنند
که فضای فرضیه نامیده می‌شود

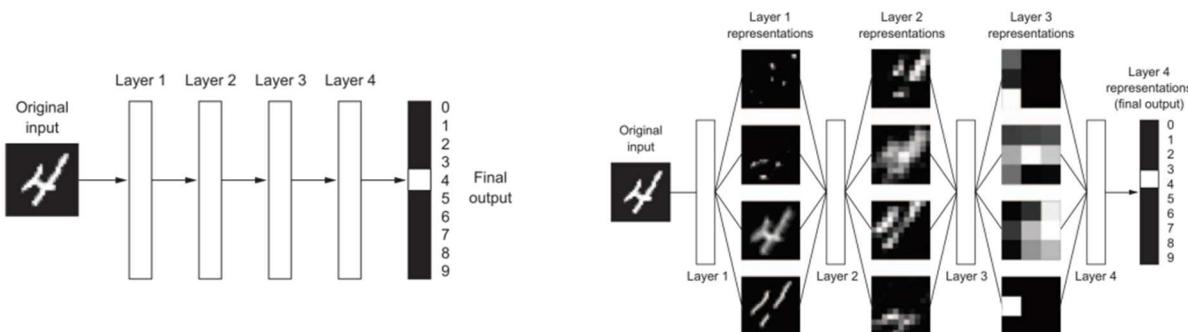


در یادگیری عمیق خود مدل بازنمایی مناسب را پیدا می‌کند

بازنمایی در یادگیری عمیق

- الگوریتم‌های یادگیری عمیق به صورت لایه به لایه و تدریجی بازنمایی‌های سطح بالا را از داده ورودی استخراج می‌کنند

- این بازنمایی‌ها به تدریج از داده ورودی فاصله می‌گیرند و هم‌زمان مناسب‌تر برای نتیجه نهایی می‌شوند



The most important difference between machine learning and deep learning is the way they learn. Machine learning algorithms typically learn by being explicitly programmed with features to extract from the data. Deep learning algorithms, on the other hand, learn by automatically extracting features from the data using artificial neural networks.

Another key difference is the type of data that machine learning and deep learning algorithms are typically used for. Machine learning algorithms are often used for tasks with structured data, such as predicting customer churn or classifying fraud. Deep learning algorithms are often used

for tasks with unstructured data, such as image recognition, natural language processing, and speech recognition.

Here is a table that summarizes the key differences between machine learning and deep learning:

Characteristic	Machine learning	Deep learning
Learning approach	Explicitly programmed with features to extract from the data	Automatically extracts features from the data using artificial neural networks
Type of data typically used for	Structured data	Unstructured data
Examples of tasks	Predicting customer churn, classifying fraud	Image recognition, natural language processing, speech recognition

Here is an example to illustrate the difference between machine learning and deep learning:

Imagine you are developing a machine learning algorithm to predict whether a customer will churn (cancel their subscription). You might start by extracting features from the customer data, such as their age, gender, subscription history, and usage patterns. You would then train a machine learning algorithm on this data to predict whether each customer is likely to churn.

To develop a deep learning algorithm for the same task, you would simply provide the algorithm with the raw customer data. The algorithm would then automatically extract features from the data and learn to predict churn patterns.

Deep learning algorithms are often more powerful than machine learning algorithms, but they can also be more complex and computationally expensive to train. Additionally, deep learning algorithms require large amounts of data to train effectively.

به نظر من، مهم‌ترین تفاوت بین یادگیری ماشین و یادگیری عمیق در پیچیدگی و عمق مدل‌های است که آن‌ها به کار می‌گیرند، که منجر به تفاوت‌هایی در انواع مشکلاتی که برای حل آنها مناسب هستند و ماهیت داده‌های مورد نیاز برای آموزش است. در اینجا تجزیه و تحلیل دقیق تر است:

۱. پیچیدگی و عمق مدل:

- یادگیری ماشینی معمولاً شامل توسعه واستفاده از الگوریتم‌های است که می‌توانند از آنها یاد بگیرند و بر اساس داده‌ها پیش‌بینی یا تصمیم‌گیری کنند. این الگوریتم‌ها اغلب به تکنیک‌های آماری برای شناسایی الگوهای روابط درون داده‌ها متکی هستند. مدل‌های یادگیری ماشینی در مقایسه با مدل‌های یادگیری عمیق، پیچیدگی کمتری دارند، با پارامترهای کمتر و معماری‌های کم عمق‌تر.

- یادگیری عمیق، از سوی دیگر، زیرمجموعه ای تخصصی از یادگیری ماشین است که از شبکه های عصبی مصنوعی با لایه های متعدد استفاده می کند و به آنها امکان می دهد تا نمایش های پیچیده ای از داده ها را بیاموزند. مدل های یادگیری عمیق می توانند به طور خودکار الگوها و ویژگی های پیچیده را از داده های خام کشف کنند، و آنها را به ویژه برای کارهایی که شامل مجموعه داده های بزرگ و با ابعاد بالا هستند، مناسب می سازد.

۲. داده های مورد نیاز:

- مدل های یادگیری ماشینی اغلب می توانند با مجموعه داده های کوچک تر عملکرد خوبی داشته باشند و برای مشکلاتی که ویژگی های ورودی نسبتاً به خوبی تعریف شده اند و خیلی پیچیده نیستند، مناسب هستند. آنها می توانند انواع مختلفی از وظایف، از جمله طبقه بندی، رگرسیون، خوش بندی، و سیستم های توصیه وغیره را انجام دهند.

- مدل های یادگیری عمیق، به دلیل افزایش پیچیدگی و عمق، اغلب به حجم زیادی از داده ها برای آموزش نیاز دارند. آنها در کارهای که شامل داده های ساختار نیافرته مانند تصاویر، صدا و متن است، برتری دارند، جایی که می توانند به طور خودکار نمایش های سلسله مراتبی داده ها را بیاموزند و ویژگی های مرتبط را در سطوح مختلف انتخاب استخراج کنند.

۳. مهندسی ویژگی:

- در یادگیری ماشینی سنتی، مهندسی ویژگی نقش مهمی ایفا می کند و به دانش و تخصص دامنه نیاز دارد تا ویژگی های مربوطه را از داده ها به صورت دستی استخراج کند. این فرآیند می تواند زمان بر باشد و ممکن است به شدت بر عملکرد مدل تأثیر بگذارد.

- مدل های یادگیری عمیق، با توانایی خود در یادگیری خودکار نمایش های سلسله مراتبی، اغلب نیاز به مهندسی ویژگی های دستی گسترشده را از بین می بند. آنها می توانند ویژگی های پیچیده را از داده های خام بیاموزند، وابستگی به ویژگی های دست ساز را کاهش دهند و روند کلی مدل سازی را ساده کنند.

۴. پیچیدگی محاسباتی:

- مدل های یادگیری عمیق از نظر محاسباتی فشرده هستند و به قدرت پردازش قابل توجهی نیاز دارند و اغلب نیاز به استفاده از سخت افزارهای تخصصی مانند GPU و TPU دارند. آموزش مدل های یادگیری عمیق می تواند وقت گیر و منابع فشرده باشد، به ویژه برای مجموعه داده های در مقیاس بزرگ.

- الگوریتم های یادگیری ماشین سنتی، به دلیل سادگی نسبی شان، اغلب از نظر محاسباتی کمتر نیاز دارند و می توانند برای طیف گسترده ای از مشکلات در سخت افزار محاسباتی استاندارد اعمال شوند.

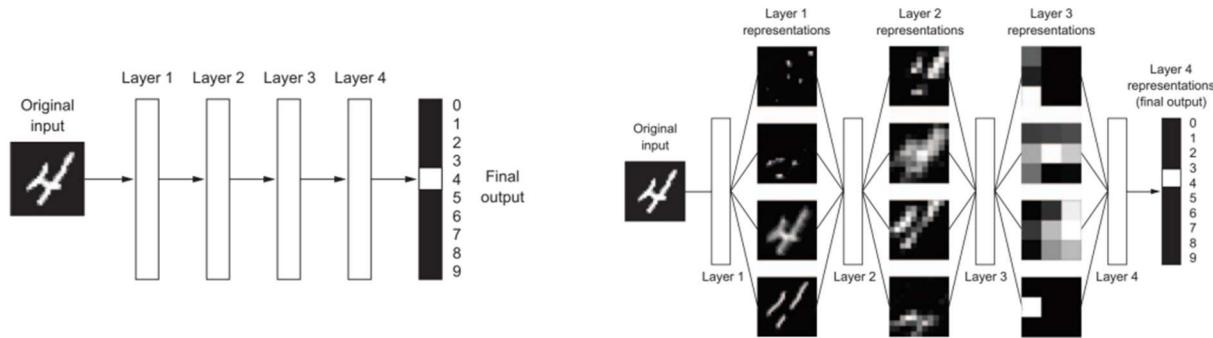
درک این تفاوت ها در تعیین این که از کدام رویکرد برای انواع مختلف مسائل استفاده شود، با در نظر گرفتن عواملی مانند در دسترس بودن داده ها، منابع محاسباتی و پیچیدگی روابط درون داده ها، حیاتی است. یادگیری ماشین و یادگیری عمیق هر دو نقاط قوت و کاربردهای خود را دارند و انتخاب رویکرد مناسب به نیازهای خاص مشکل در دست بستگی دارد.

(ب)



- الگوریتم‌های یادگیری عمیق به صورت لایه به لایه و تدریجی بازنمایی‌های سطح بالا را از داده ورودی استخراج می‌کنند

- این بازنمایی‌ها به تدریج از داده ورودی **فاصله می‌گیرند و هم‌زمان مناسب‌تر برای نتیجه نهایی می‌شوند**



به طور کلی لایه یازدهم یک شبکه یادگیری عمیق ۱۶ لایه برای رسیدن به نتیجه نهایی در طبقه بندی مناسب تر از لایه ۷ است. این به این دلیل است که لایه یازدهم به لایه خروجی شبکه نزدیکتر است و بنابراین فرصت بیشتری برای یادگیری الگوهای پیچیده در داده داشته است.

لایه‌های اولیه یک شبکه یادگیری عمیق معمولاً ویژگی‌های ساده مانند لبه‌ها و اشکال در تصاویر را یاد می‌گیرند. لایه‌های بعدی شبکه ویژگی‌های پیچیده‌تری را یاد می‌گیرند، مانند قطعات شی و کل اشیاء. بنابراین، لایه‌های بعدی شبکه بهتر می‌توانند ویژگی‌های سطح بالایی را که برای طبقه‌بندی دقیق مورد نیاز است، نشان دهند.

با این حال، مواردی وجود دارد که لایه هفتم یک شبکه یادگیری عمیق ممکن است برای طبقه بندی مناسب تر از لایه یازدهم باشد. به عنوان مثال، اگر شبکه بر روی یک مجموعه داده کوچک آموخته داده می‌شود، یا اگر مجموعه داده بسیار نویز باشد، لایه‌های قبلی شبکه ممکن است کمتر مستعد بیش از حد برازش باشند. علاوه بر این، اگر کار نسبتاً ساده باشد، مانند طبقه‌بندی تصاویر ارقام دست‌نویس، لایه‌های اولیه شبکه ممکن است کاف باشد.

در نهایت، بهترین راه برای تعیین اینکه کدام لایه از یک شبکه یادگیری عمیق برای طبقه بندی مناسب تر است، آزمایش است. می‌توانید از لایه‌های مختلف شبکه به عنوان ورودی طبقه بندی کننده خود استفاده کنید و ببینید کدام لایه بهترین عملکرد را دارد.

در اینجا چند فاکتور اضافی وجود دارد که باید هنگام انتخاب لایه‌ای از یک شبکه یادگیری عمیق برای طبقه بندی استفاده کرد:

- اندازه و پیچیدگی مجموعه داده: مجموعه داده‌های کوچکتر و ساده‌تر ممکن است از استفاده از لایه‌های قبلی شبکه بهره مند شوند.
- کار در دست: کارهای پیچیده تر ممکن است نیاز به استفاده از لایه‌های بعدی شبکه داشته باشد.
- خطر بیش از حد برازش: لایه‌های اولیه شبکه ممکن است کمتر مستعد بیش از حد برازش باشند، به خصوص در مجموعه داده‌های کوچک یا پرسرو صدا.
- هزینه محاسباتی: آموخته یک طبقه بندی کننده در لایه‌های بعدی شبکه می‌تواند از نظر محاسباتی گران‌تر باشد.

In this scenario, while the specific suitability of the 7th layer versus the 11th layer of the deep learning network for achieving the final classification result depends on various factors, it is essential to consider several key points for analysis:

1. Information Representation:

- The 7th layer, being closer to the input layer, likely captures more low-level and simpler features and patterns from the raw data. These features might include basic shapes or textures that can aid in distinguishing fundamental characteristics of the input data.
- The 11th layer, being deeper in the network, is expected to capture more abstract and high-level representations. It may identify complex patterns, combinations of features, or more sophisticated relationships among the features learned from the previous layers.

2. Complexity of Task and Data:

- If the classification task is relatively straightforward and can be adequately represented by simpler features, the 7th layer might be sufficient for achieving the final classification. This is especially true if the data does not have intricate or deeply embedded patterns that require complex hierarchical representations.

- However, if the classification task is complex and involves intricate relationships among different features in the data, the 11th layer might be more suitable. It can leverage the deep representations learned from the preceding layers to capture the more intricate and abstract features necessary for accurate classification.

3. Overfitting and Regularization:

- Deeper layers can be prone to overfitting, especially if the dataset is not large enough or if the network architecture is overly complex. In such cases, the 7th layer might be a better choice as it could be less prone to overfitting and could provide a more generalized representation of the data for classification.

- Regularization techniques such as dropout or batch normalization may have been applied to prevent overfitting in deeper layers. These techniques can make the 11th layer more suitable for the final result by ensuring that the network generalizes well to unseen data.

4. Computational Efficiency:

- Training and evaluating deeper layers can be more computationally intensive and time-consuming compared to shallower layers. If the time and computational resources are limited, the 7th layer might be preferred as it requires less computation and can provide reasonably accurate results without the added complexity of the deeper layers.

Considering these points, it is crucial to evaluate the specific characteristics of the data, the complexity of the classification task, and the architecture and training of the deep learning network to determine whether the 7th or the 11th layer is more suitable for reaching the final classification result. Both layers play distinct roles in extracting features and representations, and their suitability depends on the unique demands of the given task and data.

ج)

پرسپترون چندلایه

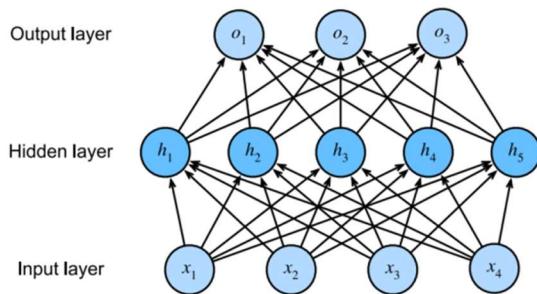
- یک شبکه MLP با تابع فعال‌سازی مناسب در لایه‌های میانی و تعداد نورون کافی می‌تواند هر تابعی را پیاده‌سازی کند

- اصطلاحاً Universal Approximator است

- حتی فقط با یک لایه میانی با تعداد نورون به اندازه کافی زیاد

- قسمت سخت کار آموزش وزن‌های مدل است

- برای تقریب بسیاری از توابع می‌توانیم از شبکه‌های عمیق تر (بجای عریض تر) استفاده کنیم که بسیار فشرده‌تر و کارآتر باشد



به طور کلی، شبکه‌های عمیق تر برای تقریبی توابع کارآمدتر از شبکه‌های عریض تر هستند. این به این دلیل است که شبکه‌های عمیق تر می‌توانند الگوهای پیچیده تری را در داده‌ها با پارامترهای کمتر یاد بگیرند.

برای تقریب یک تابع، یک شبکه عصبی نیاز به یادگیری نگاشت از فضای ورودی به فضای خروجی دارد. این نگاشت می‌تواند پیچیده باشد، به خصوص برای توابعی که بسیار غیرخطی هستند. شبکه‌های عمیق تر می‌توانند نگاشتهای پیچیده را با تجزیه به نگاشتهای ساده تر یاد بگیرند. هر لایه از یک شبکه عمیق، نگاشت ساده تری را می‌آموزد، و لایه‌های بعدی یاد می‌گیرند که این نگاشتهای گونه‌ای ترکیب کنند که تابع هدف را تقریبی کند.

از سوی دیگر، شبکه‌های عریض تر، با افزایش تعداد پارامترها در هر لایه، نگاشتهای پیچیده تری را یاد می‌گیرند. این می‌تواند کارایی کمتری داشته باشد، به خصوص برای توابعی که خیلی غیرخطی نیستند.

علاوه بر این، شبکه‌های عمیق تر در برابر نویز در داده‌ها مقاوم تر هستند. این به این دلیل است که شبکه‌های عمیق تر می‌توانند یادگیری نمایش الگوهای اساسی در داده‌ها، فیلتر کردن نویز را بیاموزند. از سوی دیگر، شبکه‌های عریض تر، بیشتر در معرض نویز در داده‌ها هستند.

در اینجا چند نمونه از مواردی که شبکه‌های عمیق تر کارآمدتر از شبکه‌های عریض تر هستند آورده شده است:

- تشخیص تصویر: شبکه‌های عمیق تر به نتایج پیشرفته‌تری در کارهای تشخیص تصویر، مانند طبقه‌بندی تصاویر اشیا و چهره‌ها دست یافته‌اند.
- پردازش زبان طبیعی: شبکه‌های عمیق تر به نتایج پیشرفته‌تری در وظایف پردازش زبان طبیعی، مانند ترجمه ماشینی و خلاصه‌سازی متن، دست یافته‌اند.
- تشخیص گفتار: شبکه‌های عمیق تر به نتایج پیشرفته‌ای در زمینه وظایف تشخیص گفتار دست یافته‌اند.

با این حال، مواردی وجود دارد که شبکه‌های عریض تر ممکن است کارآمدتر از شبکه‌های عمیق تر باشند. به عنوان مثال، اگر تابع بسیار ساده است، یا اگر مجموعه داده بسیار کوچک است، یک شبکه عریض تر ممکن است بتواند تابع را با پارامترهای کمتری نسبت به یک شبکه عمیق تر یاد بگیرد. علاوه بر این، شبکه‌های عریض تر می‌توانند برای آموزش انواع خاصی از ساخت افزار کارآمدتر باشند.

در نهایت، بهترین راه برای تعیین اینکه آیا باید از یک شبکه عمیق تر یا یک شبکه عریض تر برای تقریب یک تابع استفاده کرد، آزمایش است. می‌توانید از معماری‌های مختلف شبکه استفاده کنید و ببینید کدام معماری در مجموعه داده شما بهترین عملکرد را دارد.

The choice between using deeper networks or wider networks to approximate functions depends on various factors, including the complexity of the function being approximated, the availability of data, computational resources, and the specific requirements of the task at hand. Here's a detailed analysis of the efficiency considerations for both deeper and wider networks:

1. Deeper Networks:

- **Advantages:**

- Deeper networks have the ability to capture more intricate and abstract features and relationships within the data. They excel at learning hierarchical representations of the data, enabling them to model complex functions and extract high-level features.
- They can learn to represent the data in a more efficient and compact manner by sequentially composing multiple non-linear transformations.

- **Challenges:**

- Training deeper networks can be more challenging, as they are susceptible to issues such as vanishing or exploding gradients, which can hinder convergence and make training more complex and time-consuming.
- Deeper networks often require more data for training to prevent overfitting and to ensure that the learned representations generalize well to unseen data.

2. Wider Networks:

- **Advantages:**

- Wider networks, with a larger number of neurons in each layer, can capture more diverse features simultaneously. They are effective at learning broad and varied representations of the data, which can be beneficial for tasks where capturing diverse features is crucial.
- They can be computationally more efficient than deeper networks, as they often require less time for training due to the simpler structure and less severe vanishing gradient problem.

- **Challenges:**

- Wider networks may be more prone to overfitting, especially when the available data is limited. Regularization techniques are often necessary to prevent overfitting and ensure the model's generalizability.
- Scaling up the width of the network significantly can lead to increased computational and memory requirements, making them more resource-intensive.

In practice, there is no one-size-fits-all answer to whether deeper or wider networks are more efficient for function approximation. The choice often depends on the specific characteristics of the data, the complexity of the function being approximated, the availability of data, and the computational resources at hand. A combination of both strategies, known as 'residual networks,' has often been successful in achieving better performance by combining the benefits of both deeper and wider networks. The choice between deeper and wider networks is a crucial consideration in designing efficient and effective models for various machine learning tasks.

(د) مزایای افزودن لایه های بیشتر به شبکه عصبی عمیق:

- افزایش ظرفیت برای یادگیری الگوهای پیچیده: شبکه های عمیق تر دارای پارامترهای بیشتری هستند که به آنها امکان می دهد الگوهای پیچیده تری را در داده ها یاد بگیرند. این می تواند برای کارهایی مانند تشخیص تصویر، پردازش زیان طبیعی و تشخیص گفتار مفید باشد.
- بهبود عملکرد تعمیم: شبکه های عمیق تر می توانند بهتر از شبکه های کم عمق به داده های غیر قابل مشاهده تعمیم دهنند. این به این دلیل است که شبکه های عمیق تر می توانند نمایش های انتزاعی بیشتری از داده ها را بیاموزند.
- افزایش استحکام در برابر نویز: شبکه های عمیق تر نسبت به شبکه های کم عمق در برابر نویز در داده ها مقاوم تر هستند. این به این دلیل است که شبکه های عمیق تر می توانند با یادگیری نمایش الگوهای اساسی در داده ها، فیلتر کردن نویز را بیاموزند.

معایب افزودن لایه های بیشتر به شبکه عصبی عمیق:

- افزایش هزینه محاسباتی: شبکه های عمیق تر به منابع محاسباتی بیشتری برای آموزش و استقرار نیاز دارند. این می تواند یک عامل محدود کننده برای کارهایی باشد که نیاز به عملکرد در زمان واقعی دارند.
- افزایش خطر بیش برازش: شبکه های عمیق تر نسبت به شبکه های کم عمق بیشتر در معرض خطر بیش برازش قرار دارند. این به این دلیل است که شبکه های عمیق تر دارای پارامترهای بیشتری هستند که به آنها فرصت بیشتری برای یادگیری داده های آموزشی می دهد.
- مشکل گرادیان ناپدید شدن یا گرادیان های انفجاری: مشکل گرادیان ناپدید شدن یا گرادیان های انفجاری می تواند هنگام آموزش شبکه های عصبی عمیق رخ دهد. این مشکل زمانی رخ می دهد که با عمیق تر شدن شبکه، گرادیان های تابع خطای نسبت به وزن های شبکه بسیار کوچک یا بسیار بزرگ می شوند. این می تواند آموزش موثر شبکه را دشوار کند.

به طور کلی، اضافه کردن یا عدم افزودن لایه های بیشتر به یک شبکه عصبی عمیق به وظیفه خاص و منابع موجود بستگی دارد. اگر کار پیچیده است و به دقت بالایی نیاز دارد، یک شبکه عمیق تر ممکن است سودمند باشد. با این حال، اگر کار ساده باشد یا اگر منابع محاسباتی محدود باشد، ممکن است یک شبکه کم عمق تر مناسب تر باشد.

در اینجا چند نکته برای افزودن موثر لایه های بیشتر به شبکه عصبی عمیق وجود دارد:

- از تکنیک های منظم سازی برای جلوگیری از برازش بیش از حد استفاده کنید.
- برای تنظیم میزان یادگیری در طول آموزش از یک زمانبندی نرخ یادگیری استفاده کنید.
- از تکنیک هایی مانند نرمال سازی دسته ای و تنظیم اولیه وزن برای بهبود پایداری تمرین استفاده کنید.
- از آموزش انتقال(Transfer Learning) برای مقداردهی اولیه وزن های شبکه با وزنه هایی که برای یک کار مشابه از قبل آموزش دیده اند استفاده کنید.

Adding more layers to a deep neural network can offer several advantages in terms of learning complex representations of data, but it also introduces certain challenges and complexities that need to be carefully considered. Here's a detailed analysis of the advantages and disadvantages of increasing the depth of a neural network:

Advantages:

1. **Hierarchy of Features:** Deeper networks can learn to represent data in terms of multiple levels of abstraction. Each layer captures increasingly complex features, allowing the network to learn intricate patterns and relationships within the data.
2. **Improved Representation Learning:** With more layers, the network can discover and represent more abstract and nuanced features from the data, enabling it to extract high-level information that may be crucial for complex tasks such as image recognition, natural language processing, and speech recognition.
3. **Enhanced Model Capacity:** Deeper networks have a greater capacity to model complex functions and data distributions, enabling them to handle more intricate tasks and datasets. They can learn to approximate highly non-linear and intricate mappings between inputs and outputs.

Disadvantages:

1. **Vanishing and Exploding Gradients:** Deeper networks are susceptible to the vanishing gradient problem, where the gradients shrink exponentially as they propagate back through the layers during training. This can hinder the convergence of the network and make it difficult to train effectively.
2. **Computational Complexity:** Deeper networks are computationally more intensive, requiring increased processing power and memory. Training such networks can be time-consuming and resource-intensive, especially without access to specialized hardware like GPUs or TPUs.
3. **Overfitting and Generalization:** Deep networks with many layers may have a higher risk of overfitting, especially when the dataset is limited. Regularization techniques such as dropout, batch normalization, or early stopping may be required to prevent overfitting and ensure the model generalizes well to unseen data.
4. **Architectural Complexity:** Deeper networks introduce more hyperparameters, making the overall architecture more complex and challenging to optimize. Choosing the appropriate architecture, activation functions, and optimization algorithms becomes more critical and may require extensive experimentation and tuning.

Balancing these advantages and disadvantages is crucial in designing an effective deep neural network. Proper regularization, optimization techniques, and careful tuning of hyperparameters can help mitigate the challenges associated with deeper networks while harnessing their capacity to learn complex representations from data.

منابع:

<https://bard.google.com/>

<https://chat.openai.com/>

<https://claude.ai/chats>

لینک چت من با :chat gpt

<https://chat.openai.com/share/b37d975d-0840-44be-8dee-202c7ef9ca63>

پایان