

ملیکا محمدی فخار - ۹۹۵۲۲۰۸۶
 «در انجام این تمرین علاوه بر اطلاعات اینجانب و اسلایدهای درس تنها از سایت
<https://chat.openai.com> به عنوان منبع کمکی استفاده شده است.»

۱.

Subject: _____ Year: _____ Month: _____ Date: _____

Subject: _____ Year: _____ Month: _____ Date: _____

$$P(w_i | \text{class}) = \frac{\text{freq}(w_i, \text{class})}{N_{\text{class}}} \quad \text{class} \in \{0, 1\} \quad (1) \text{ الف}$$

Word	C_0	C_1	C_0	C_1
فشاری	۲	۱	$\frac{2}{9}$	$\frac{1}{8}$
نرمشی	۲	۱	$\frac{2}{9}$	$\frac{1}{8}$
علمی	۲	۱	$\frac{2}{9}$	$\frac{1}{8}$
استفادی	۱	۱	$\frac{1}{9}$	$\frac{1}{8}$
اجتماعی	۱	۲	$\frac{1}{9}$	$\frac{2}{8}$
سیاسی	۱	۲	$\frac{1}{9}$	$\frac{2}{8}$
N_{class}	۹	۸		

Probabilities →

Likelihood score: $\prod_{i=1}^m \frac{P(w_i | 0)}{P(w_i | 1)} = \frac{\frac{2}{9} \times \frac{2}{9} \times \frac{2}{9} \times \frac{1}{9}}{\frac{1}{8} \times \frac{1}{8} \times \frac{1}{8} \times \frac{1}{8}} > 1$ ← ستمی به کلاس ۰

(-) Laplace Smoothing استفاده می‌کنیم:

$$P(w_i | \text{class}) = \frac{\text{freq}(w_i, \text{class}) + \alpha}{N_{\text{class}} + \alpha k}$$

→ num of unique vocabs = ۷

$$\prod_{i=1}^m \frac{P(w_i | 0)}{P(w_i | 1)} = \frac{\frac{3}{17} \times \frac{3}{17} \times \frac{3}{17} \times \frac{1}{17}}{\frac{2}{10} \times \frac{2}{10} \times \frac{2}{10} \times \frac{1}{10}} > 1 \quad \text{ستیمی به کلاس ۰}$$

((گزینه))

CS Scanned with CamScanner

۲.

۳.

Subject:

Year: Month: Date:

Subject:

Year: Month: Date:

Probit: $\varphi(a) = \int_{-\infty}^a N(\theta | 0, 1) d\theta$ (۳)

نوع نرمال

اگر \vec{x} بردار ورودی، \vec{w} بردار وزن ها، b بایاس، y خروجی این است:

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

تابع φ به ازای یک نمونه (\vec{x}_i, y_i) به صورت زیر است:

$$P(y_i | \vec{x}_i, \vec{w}, b) = y_i \cdot \varphi(\vec{x}_i \cdot \vec{w} + b) + (1 - y_i) \cdot (1 - \varphi(\vec{x}_i \cdot \vec{w} + b))$$

حال به کمک loss function می توانیم (منبع یادگیری):

$$L_i(\vec{w}, b) = -\log(P(y_i | \vec{x}_i, \vec{w}, b))$$

$$L_i(\vec{w}, b) = \begin{cases} -\log(\varphi(\vec{x}_i \cdot \vec{w} + b)) & y_i = 1 \\ -\log(1 - \varphi(\vec{x}_i \cdot \vec{w} + b)) & y_i = 0 \end{cases}$$

$$\Rightarrow L_i(\vec{w}, b) = -y_i \log(\varphi(\vec{x}_i \cdot \vec{w} + b)) - (1 - y_i) \log(1 - \varphi(\vec{x}_i \cdot \vec{w} + b))$$

حال برای به دست آوردن total loss، از loss هر نمونه می بینیم:

$$L(\vec{w}, b) = \frac{1}{n} \sum_{i=1}^n L_i(\vec{w}, b) \rightarrow \text{میانگین}$$

میانگین

۴.
الف.

اضافه کردن ویژگی **non-linearity**: توابع فعال سازی **non-linearity** به شبکه وارد می کنند و این امکان را فراهم می کنند که شبکه، الگوهای پیچیده در داده ها را مدل سازی کند. بدون این ویژگی، شبکه به عنوان یک تبدیل خطی عمل می کند و قادر به یادگیری و نمایش الگوهای پیچیده در داده ها نمی شود.

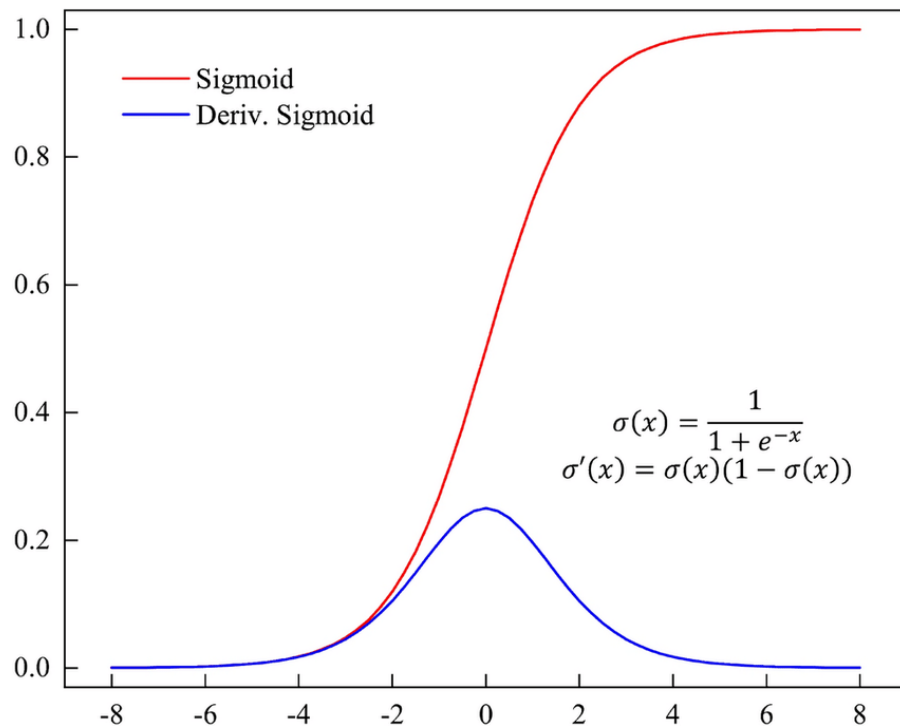
امکان استفاده از گرادیان در **back propagation**: توابع فعال سازی به الگوریتم برگشت به عقب کمک می کنند که برای آموزش شبکه های عصبی استفاده می شود. به طوری که با مشتق گیری از آنها، به روزرسانی وزن های شبکه از طریق کاهش گرادیان ممکن می شود. گرادیان توابع فعال سازی بر روی سرعت و کیفیت همگرایی (**convergence**) اثرگذار است.

ب.
با اینکه در تئوری هر تابع غیرخطی می تواند به عنوان یک تابع فعال سازی شناخته شود اما در واقعیت بستگی به مسئله دارد زیرا هر تابع فعال سازی (اگر چه غیرخطی باشد) برای هر مسئله ای مناسب نیست.
برای مثال ممکن است برخی از آنها به لحاظ محاسباتی بهینه نباشند و یا مدل را دچار مشکلاتی از قبیل **vanishing gradient** کنند. از آن جا که بسیاری از مسائل با استفاده از گرادیان حل می شوند، تابع فعال سازی بایست مشتق پذیر نیز باشد. همچنین بایست بازه خروجی مناسبی برای حل مسئله داشته باشد. مثلاً برای حل مسئله دو کلاسه خروجی بهتر است بین ۰ و ۱ باشد در غیر این صورت نیاز به یک مرحله نرمال سازی اضافه داریم.

۵.

(الف)

تابع sigmoid:



مزایا:

خروجی این تابع فعال‌سازی بین ۰ و ۱ است، از این رو انتخاب مناسبی برای مسائل **binary classification** می‌باشد. همچنین برای مدل‌های احتمالاتی (از آن‌جا که احتمال همواره بین ۰ و ۱ است) نیز مناسب است. همچنین این تابع غیرخطی، پیوسته و همواره مشتق‌پذیر است بنابراین با الگوریتم‌های **gradient-based** مانند **gradient descent** که با استفاده از مشتق‌گیری وزن‌ها را آپدیت می‌کنند سازگار است.

معایب:

مقدار مشتق این تابع کوچک است به طوری که حداکثر به ۰.۲۵ می‌رسد که این موضوع باعث کندی روند یادگیری می‌شود. همچنین هرچه ورودی این تابع از ۰ فاصله بگیرد (در دو جهت)، مشتق آن کوچک‌تر می‌شود و به صفر نزدیک می‌شود. به همین دلیل روند یادگیری در مشتق‌های بسیار

کوچک عملاً متوقف می‌شود. (یک راه حل برای این مشکل استفاده از توابع فعال‌سازی خانواده Relu است.)

میانگین مقادیر تابع sigmoid، ۰ نیست و همواره مقادیر مثبت تولید می‌کند یا به اصطلاح non zero-centered است. از آن‌جا که ورودی هر لایه خروجی لایه قبلی آن است که همیشه مثبت است، روند convergence این تابع کند می‌شود. (یک راه حل برای این مشکل استفاده از توابع فعال‌سازی مانند tanh است.) از آن‌جا که معادله این تابع exponential است محاسبات پرهزینه‌ای دارد.

تابع softmax:

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

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

• تابع ضرر متناسب با این تابع فعال‌سازی، categorical cross-entropy است

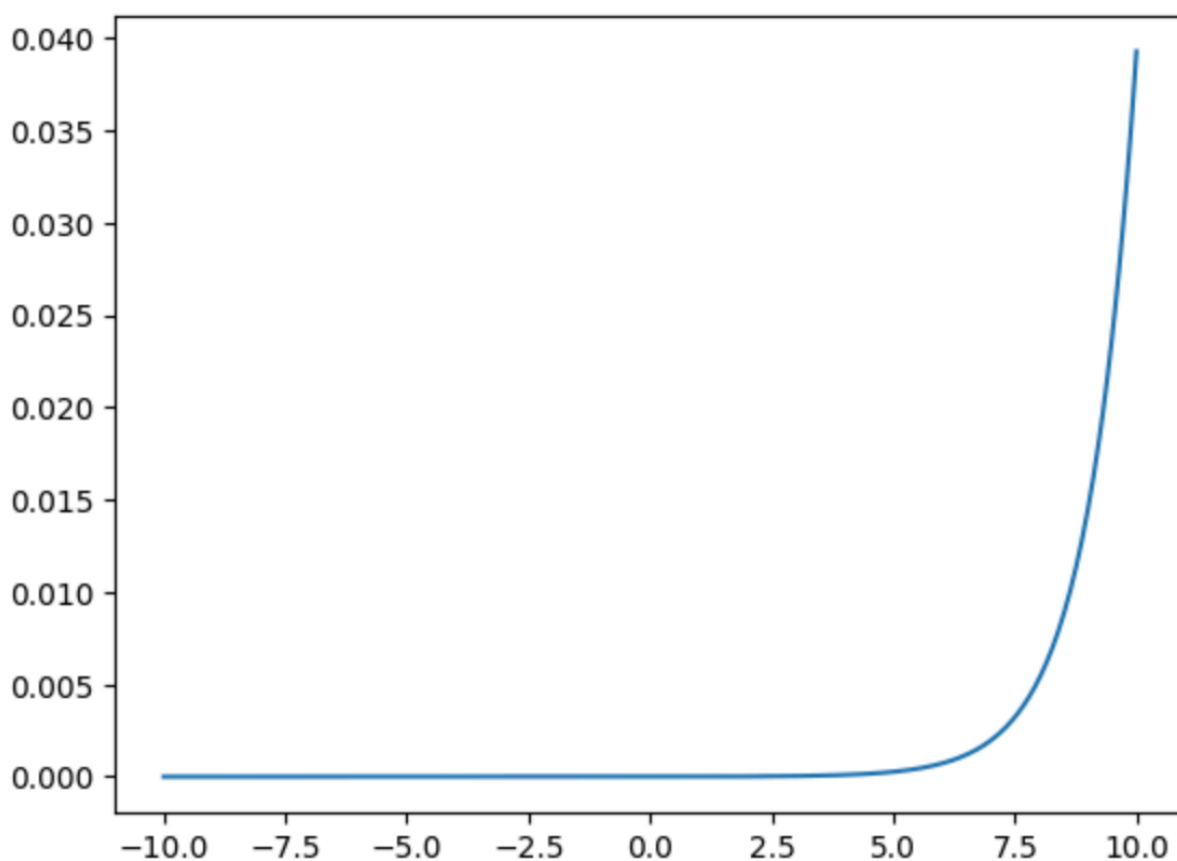
$$l(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^q y_i \log \hat{y}_i$$

مشتق softmax

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

$$= \log \sum_{k=1}^q \exp(o_k) \sum_{i=1}^q y_i - \sum_{i=1}^q y_i o_i = \log \sum_{k=1}^q \exp(o_k) - \sum_{i=1}^q y_i o_i$$

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



مزایا:
این تابع می‌تواند یک وکتور از اعداد حقیقی را به احتمالاتی تبدیل کند که مجموع آن‌ها ۱ است، بنابراین انتخاب مناسبی در حل مسائل **multiclass classification** است. از آن‌جا که **softmax** بر روی وکتور ورودی خود نرمال‌سازی انجام می‌دهد برای قرار گیری در لایه آخر مناسب است. (کلاسی که نظیر بیشترین احتمال است به عنوان کلاس تشخیص داده شده توسط مدل در نظر گرفته می‌شود).
این تابع نیز مانند **sigmoid** غیرخطی، پیوسته و همواره مشتق‌پذیر است.

معایب:

مانند sigmoid از آن جا که معادله این تابع exponential است محاسبات پرهزینه‌ای دارد.

این تابع نسبت به اسکیل ورودی حساس است و احتمالات خروجی آن به دلیل این حساسیت به شدت تغییر می‌کنند و این امر کیفیت یادگیری را کاهش می‌دهد. به مثال زیر دقت کنید:

1. Original Input Scores:

- Class A: 2.0
- Class B: 1.0
- Class C: 0.1

2. Calculate Exponentials:

- $\exp(2.0) \approx 7.39$
- $\exp(1.0) \approx 2.72$
- $\exp(0.1) \approx 1.11$

3. Normalize to Probabilities:

- Class A: $7.39 / (7.39 + 2.72 + 1.11) \approx 0.700$
- Class B: $2.72 / (7.39 + 2.72 + 1.11) \approx 0.254$
- Class C: $1.11 / (7.39 + 2.72 + 1.11) \approx 0.046$

1. Modified Input Scores (Scaled by 10):

- Class A: 20.0
- Class B: 10.0
- Class C: 1.0

2. Calculate Exponentials:

- $\exp(20.0) \approx 485165195.41$
- $\exp(10.0) \approx 22026.47$
- $\exp(1.0) \approx 2.72$

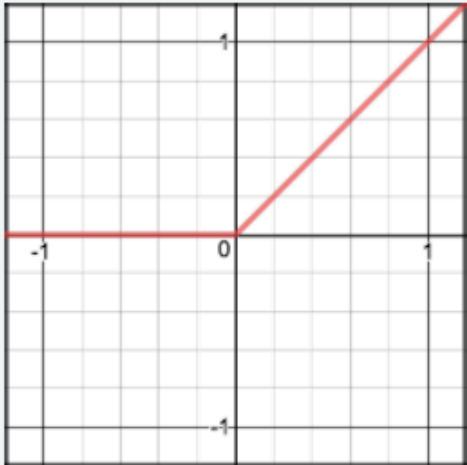
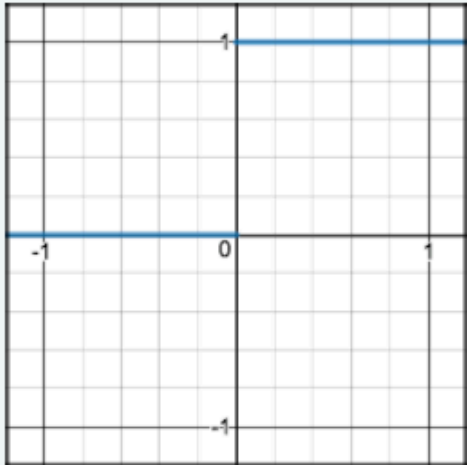
3. Normalize to Probabilities:

- Class A: $485165195.41 / (485165195.41 + 22026.47 + 2.72) \approx 0.956$
- Class B: $22026.47 / (485165195.41 + 22026.47 + 2.72) \approx 0.043$
- Class C: $2.72 / (485165195.41 + 22026.47 + 2.72) \approx 0.00000001$

برای حل مشکل این می‌توان ورودی آن را از پیش نرمال‌سازی کرد که کمی پیچیدگی محاسباتی به مدل اضافه می‌کند.

این تابع نیز مانند sigmoid، non zero-centered است.

تابع Relu:

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$
	

مزایا:

با این که این تابع برای مقادیر مثبت، خطی است با این حال با در نظر گرفتن خروجی آن به ازای مقادیر منفی یک تابع غیرخطی است.

پیاده‌سازی آن ساده است و چون از operationهای ساده‌ای استفاده می‌کند، به لحاظ محاسباتی بهینه است.

از آنجا که مقدار مشتق تابع به ازای مقادیر مثبت، برخلاف sigmoid و tanh مایل به صفر نمی‌شود، سریع‌تر همگرا می‌شود یا اصطلاحاً converge می‌کند.

خصوصیت Sparse Activation این تابع (به این معنا که تنها برخی نرون‌ها مقادیر غیرصفر تولید می‌کنند) به افزایش توانایی generalization مدل کمک می‌کند.

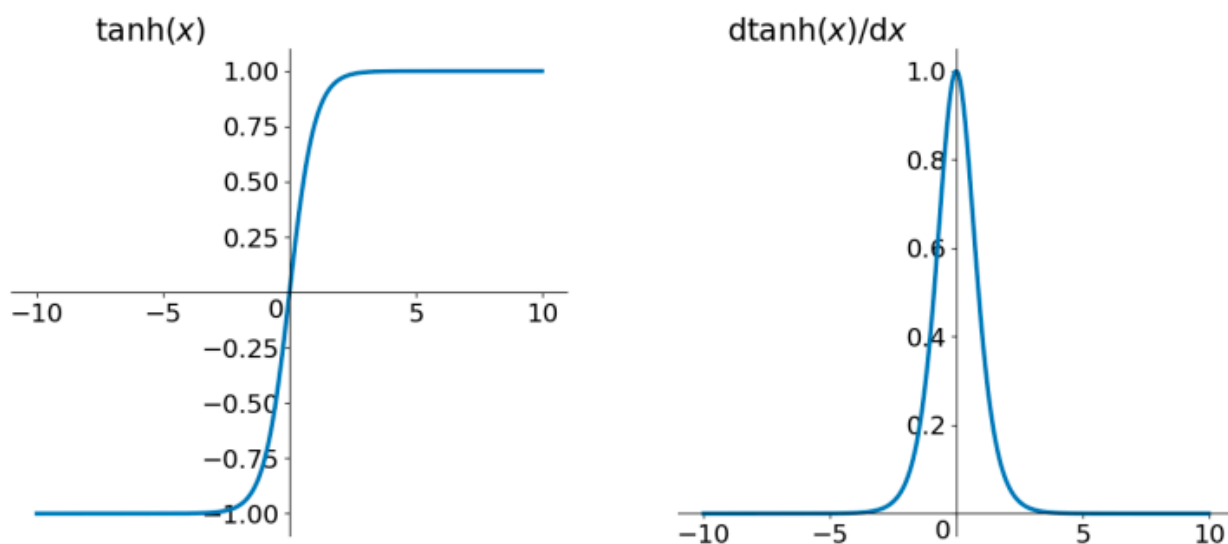
معایب:

مهم‌ترین مشکل این تابع این است که ممکن است در برخی شرایط به ازای همه ورودی‌ها خروجی ۰ دهد. در این حالت همه نرون‌ها غیرفعال می‌شوند. و یا حتی اگر همگی غیرفعال نشوند، برخی از آن‌هایی غیرفعال شوند که در حال یادگیری پترن‌های مهمی از دیتا بوده‌اند و اینگونه اطلاعات آن‌ها را از دست می‌دهیم. (برای حل این مشکل می‌توان از **leaky relu** استفاده کرد).

این تابع نیز مانند دو مورد قبلی، **non zero-centered** است.

این تابع نسبت به مقدار **learning rate** حساسیت بالایی دارد و اگر مقدار **lr** بالا باشد ممکن است نرون‌ها نقطه بهینه را رد کنند و مدل همگرا نشود.

تابع **tanh**:



مزایا:

از آنجا که خروجی تابع **tanh** بین -۱ و ۱ می‌باشد، **zero-centered** است. این تابع غیرخطی و همواره مشتق‌پذیر است.

معایب:

مانند تابع **sigmoid**، **tanh** نیز هرچه ورودی این تابع از ۰ فاصله بگیرد (در دو جهت)، مشتق آن کوچک تر می شود و به صفر نزدیک می شود. به همین دلیل روند یادگیری در مشتق های بسیار کوچک عملاً متوقف می شود. (یک راه حل برای این مشکل استفاده از توابع فعال سازی خانواده **Relu** است.)
به لحاظ محاسباتی، پرهزینه است.

ب. این فایل تکمیل و به تمرین ضمیمه شده است.
توضیح کدها به شرح زیر است:

sigmoid:

Input: The function `sigmoid(x)` takes a single input `x`, which can be a scalar, a vector, or a matrix. It can also be a PyTorch tensor.

Output: The function returns the result of applying the sigmoid function element-wise to the input `x`. The output is also a tensor with the same shape as the input.

softmax:

Input: The function `softmax(x)` takes an input vector or array `x` as its argument. This input typically represents unnormalized scores or logits for different classes.

Output: The function computes and returns the softmax probabilities as a vector of the same shape as the input `x`. Each element of the output represents the probability of the corresponding class.

Implementation:

`e_x = np.exp(x)`: The code first exponentiates each element of the input array `x` using NumPy's `np.exp` function. This step transforms the input scores into positive values.

`e_x.sum()`: It calculates the sum of the exponentiated values, which is used to normalize the probabilities.

`e_x / e_x.sum()`: The softmax probabilities are obtained by dividing each element of the exponentiated values by their sum. This step ensures that the resulting probabilities sum to 1, making them suitable for representing class probabilities.

ReLU:

Input: The function `relu(x)` takes an input vector or array `x` as its argument. This input can be a scalar, vector, or multi-dimensional array.

Output: The function applies the ReLU activation element-wise to the input and returns a new vector or array of the same shape as the input.

Implementation:

`np.maximum(0, x)`: The ReLU activation function is applied element-wise to each element of the input `x`. For each element, it replaces negative values with 0 and leaves positive values unchanged.

Leaky ReLu:

Input: The function `leaky_relu(x)` takes a PyTorch tensor `x` as its argument. This input can be a scalar, vector, or multi-dimensional tensor.

Output: The function applies the Leaky ReLU activation element-wise to the input tensor and returns a new tensor of the same shape as the input.

Implementation:

`torch.where(x > 0, x, 0.01 * x)`: The Leaky ReLU function is applied element-wise to each element of the input tensor `x`. For each element, it checks if it's greater than 0. If it is, it leaves the value unchanged. If it's less than or equal to 0, it multiplies the value by a small constant, typically 0.01.

Tanh:

Input: The function `tanh(x)` takes a PyTorch tensor `x` as its argument. This input can be a scalar, vector, or multi-dimensional tensor.

Output: The function applies the tanh activation element-wise to the input tensor and returns a new tensor of the same shape as the input.

Implementation:

`(torch.exp(x) - torch.exp(-x)) / (torch.exp(x) + torch.exp(-x))`: The tanh function is implemented using the formula that computes the hyperbolic tangent of the input. It computes $(e^x - e^{-x}) / (e^x + e^{-x})$, where e is the base of the natural logarithm.

ج.

تعداد لایه‌ها: این مدل شامل ۲ لایه **fully connected** است. برای مسائل پیچیده‌تر می‌توان از تعداد لایه‌های بیشتر استفاده کرد اما برای این مسئله با ۲ لایه هم مدل به همگرایی رسید بنابراین تعداد لایه‌ها را بیشتر نکردم.

تعداد نورون‌ها: تعداد نورون‌های لایه اول برابر تعداد ورودی است که ابعاد عکس آن را مشخص می‌کند ($3 \times 64 \times 64$)، لایه آخر نیز از آنجا که مسئله طبقه‌بندی است بایست برابر با تعداد طبقه‌ها باشد (۳)، تعداد نورون‌های لایه میانی معمولاً بین لایه قبل و بعد انتخاب می‌شود. از آنجا که مدل و مسئله تقریباً ساده بودند و نیاز به استخراج الگوهای پیچیده نبود عدد ۲۵۶ که رایج است را انتخاب کردم.

تابع فعال‌سازی: از تابع فعال‌سازی **ReLU** استفاده شده است که تعادل مناسبی بین حفظ پیچیدگی مدل (ظرفیت یادگیری آن) و بهینگی آموزش برقرار می‌کند. همچنین به لحاظ پیچیدگی محاسباتی بهینه است.

تابع ضرر: تابع ضرر **cross-entropy** معمولاً برای مسائل طبقه‌بندی چندکلاسه مناسب است. از مزایای آن می‌توان به **Logarithmic Scale** بودن و **Gradient-Based** بودن و بازه خروجی مناسب آن اشاره کرد.

د. فایل با نام **Q5.ipynp** ضمیمه شده است و توضیحات کد به شرح زیر است:

1. Custom Dataset Definition

In this section, we define a custom dataset class to load and preprocess images.

CustomDataset Class

The CustomDataset class is created to define a custom dataset by inheriting from `torch.utils.data.Dataset`.

It takes the following arguments:

`image_paths`: A list of file paths to the images.

`labels`: A list of labels corresponding to the images.

`transform`: A series of image preprocessing transformations.

`len` Method

The `__len__(self)` method returns the total number of samples in the dataset.

`getitem` Method

The `__getitem__(self, index)` method loads an image at the specified index, applies the specified transformations, and returns the image and its corresponding label.

2. Data Preparation

In this section, we prepare the data for training and evaluation.

List of Image File Paths

image_paths is a list of file paths to the image files.

Each element of the list represents a path to an image file.

These paths must be set to the actual image file paths.

Corresponding Labels

labels is a list of labels corresponding to the images in the same order as image_paths.

Each label in the list corresponds to the image with the same index in image_paths.

You can define your own labels according to your specific classification task.

Transformation

The transforms.Compose function is used to create a sequence of image transformations.

In the provided example, the transformations include:

Resizing the images to a fixed size of 64x64 pixels.

Converting the images to PyTorch tensors.

Normalizing the pixel values to be in the range [0, 1].

DataLoader

The DataLoader is used to create a batched data loader for the dataset.

It is configured with a batch size of 1, meaning one image is processed in each iteration.

The shuffle option is set to False to maintain the order of images.

3. Simple MLP Model

In this section, a simple Multi-Layer Perceptron (MLP) model is defined for image classification.

SimpleMLP Class

The SimpleMLP class is a custom class for the MLP model.

It inherits from nn.Module.

The model consists of two fully connected (linear) layers with ReLU activation in between.

The input size is specified as 64 * 64 * 3, which corresponds to a flattened image size (width x height x color channels).

The output size is set to 3, indicating the number of classes for classification.

This model is designed for classification tasks.

4. Training the Model

This section contains the code for training the MLP model.

Model Training

The model is trained for a specified number of epochs. In this example, it is set to 100.

The loss function used is the cross-entropy loss, suitable for classification tasks.

The Adam optimizer is utilized for updating the model's parameters during training.

Training Loop

The code iterates through the dataset, making forward and backward passes, and updating the model's parameters through backpropagation.

The input images are flattened with .view(images.size(0), -1) to match the input size of the model.

5. Model Evaluation

After training, the model is evaluated on the same training dataset. However, it's recommended to perform evaluation on a separate testing dataset for a more accurate assessment of the model's performance.

Model Evaluation

The model is set to evaluation mode using `model.eval()` to disable dropout and batch normalization layers.

A loop iterates through the dataset, makes predictions, and calculates accuracy.

Accuracy Calculation

The accuracy is calculated by comparing the predicted labels to the true labels and counting the correct predictions.

The final accuracy percentage is printed to the console.

۶.

با توجه به مشکل dying gradient تابع ReLU ممکن است برخی از نورون‌ها در طول فرایند آموزش غیرفعال شوند.

استفاده از یک fixed threshold به مقدار ۰.۵ ممکن است دو مشکل ایجاد کند:

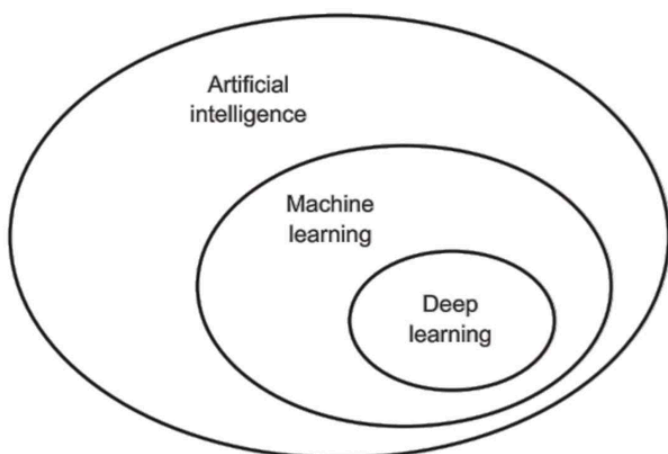
اول اینکه چون ثابت است قابل fine tune کردن نیست و اگر توزیع دیتا imbalance باشد، مناسب نیست و ممکن است به biased classification منجر شود.

دوم اینکه استفاده از threshold خاصیت احتمالی بودن (probabilistic) بودن تابع سیگموئید را از بین می‌برد که این برای فرایند آموزش مناسب نیست. (برای مثال در مرحله جریمه کردن مدل، تمام پیش‌بینی‌های اشتباه را به یک اندازه جریمه می‌کند اما اگر از خروجی سیگموئید استفاده کنیم مقدار جریمه به ازای هر اشتباه یکسان نیست.)

۷.
الف.

فکر می‌کنم مهم‌ترین تفاوت این دو در **feature extraction** و نحوه بازنمایی داده است. در ماشین لرنینگ این فیچرها به صورت دستی استخراج می‌شوند و برای پیش‌بینی به کار گرفته می‌شوند و بازنمایی قابل درکی از داده برای انسان ایجاد می‌کنند. اما این فرایند در دیپ لرنینگ توسط خود مدل انجام می‌شود و به بازنمایی‌های **high level** می‌رسد که در نظر انسان غیرقابل درک و اصطلاحاً مانند یک جعبه سیاه است.

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



ب. از آن جا که لایه ۱۱ام لایه‌ای عمیق‌تر است و بازنمایی‌اش نسبت به فیچرهای داده‌های ورودی **high-level** تر و **abstract** تر است احتمالاً از لایه ۷ام مناسب‌تر است. البته اگر تسک ما از نوعی باشد که نیاز باشد **detail** ورودی‌ها را حذف کنیم و با توجه به کلیات داده‌ها را طبقه‌بندی کنیم، لایه ۷ام می‌تواند مناسب‌تر باشد.

ج. این سوال پاسخ یکتا ندارد و وابسته به نوع مسئله است. اگر مسئله پیچیده و تعداد داده‌ها زیاد باشد استفاده از شبکه عمیق (تعداد بالای لایه‌ها) و اگر مسئله ساده‌تر و یا تعداد داده‌ها کم باشد استفاده از شبکه عرضی (تعداد بالای نوروں‌های لایه‌های میانی) مناسب‌تر است.

د. مزایا: هرچه شبکه عمیق‌تر باشد پتانسیل یادگیری بالاتری دارد. فیچرها و الگوهای سطح بالاتری از داده‌های ورودی به دست می‌آورد و بر روی تسک‌های پیچیده پرفورمنس بهتری دارد.

معایب: پیچیدگی محاسباتی افزایش می‌یابد و در نتیجه زمان آموزش طولانی‌تر می‌شود. از آن جا که توانایی یادگیری مدل افزایش یافته این احتمال وجود دارد که مدل داده‌های ورودی را حفظ کند و دچار **overfitting** شود.