

جوابیه تمرین ۴

با سلام به دانشجویان گرامی امیدوارم که حالتون خوب باشه. جواب های سوالات ۱، ۲ و ۳ تمرین ۳ با کسب اجازه از سر تدرسیار براساس جواب های گرد آوری شده دانشجویان که به این سوالات پاسخ و توضیح مناسب را داده‌اند است. جواب سوالات ۴ و ۵ نیز در ادامه قرار گرفته شده است.

سوال ۱

(الف)

هدف توابع فعال‌ساز در شبکه‌های عصبی چند لایه (MLP) این است که به مدل قابلیت یادگیری روابط غیرخطی را بدهند. توابع فعال‌ساز اعمال غیرخطی را به مدل اضافه می‌کنند که باعث می‌شود شبکه بتواند الگوهای پیچیده‌تری را یاد بگیرد. بدون استفاده از توابع فعال‌ساز، شبکه‌های عصبی فقط قادر به یادگیری توابع خطی خواهند بود.

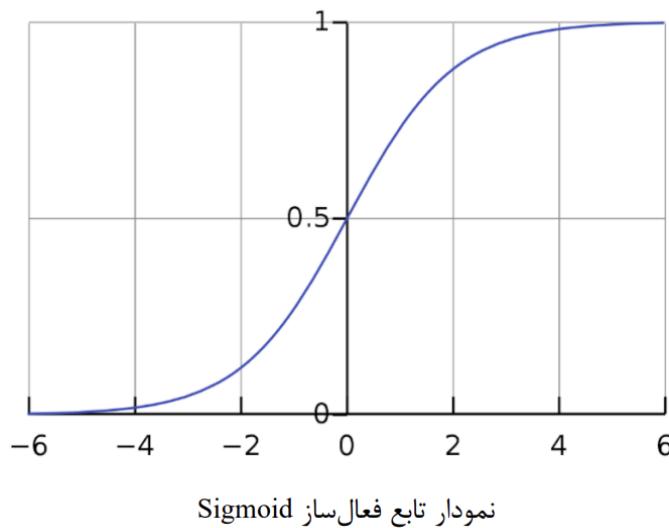
اگر در یک شبکه MLP از توابع فعال‌ساز استفاده نشود، دو مشکل اساسی پیش خواهد آمد:

۱. عدم توانایی در یادگیری روابط غیرخطی : هر لایه در یک شبکه عصبی خطی یک ترکیب خطی از ورودی‌هایش تولید می‌کند. اگر تابع فعال‌ساز غیرخطی استفاده نشود، ترکیب لایه‌های خطی متواالی همچنان یک ترکیب خطی خواهد بود. به عبارت دیگر، کل شبکه مانند یک لایه خطی واحد عمل می‌کند و نمی‌تواند روابط غیرخطی بین ورودی‌ها و خروجی‌ها را مدل‌سازی کند.

۲. کاهش قدرت شبکه : توانایی شبکه‌های عصبی در حل مسائل پیچیده و متنوع به دلیل وجود توابع فعال‌ساز غیرخطی است. بدون تابع فعال‌ساز، شبکه قدرت کافی برای انجام تفکیک و طبقه‌بندی داده‌های پیچیده را نخواهد داشت و عملکرد آن مشابه رگرسیون خطی خواهد بود.

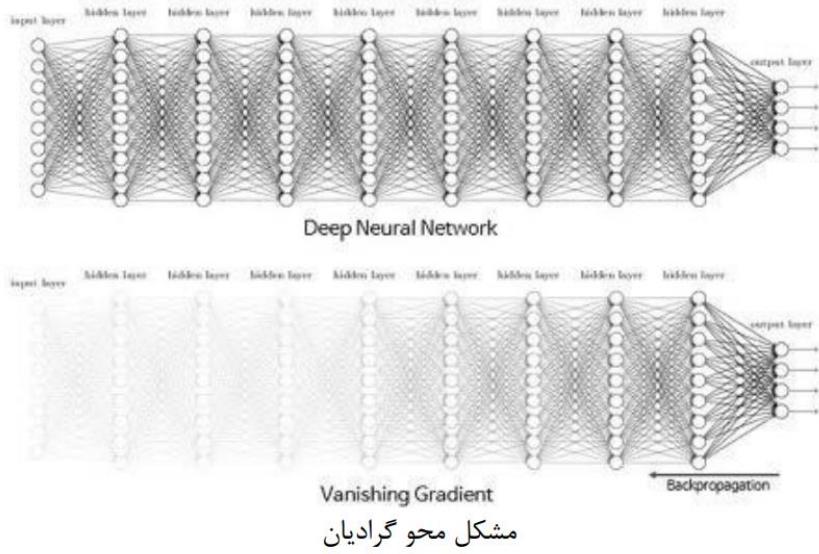
(ب)

تابع فعال‌ساز Sigmoid که با $\sigma(x) = \frac{1}{1+e^{-x}}$ فرموله می‌شود، به منحنی S مانندی که دارد معروف بوده به این معنی که در دو انتهای صاف و در وسط شیب دار است و به دلیل گرادیان ملایم و محدوده خروجی بین ۰ و ۱، یک تابع فعال‌ساز پرکاربرد در شبکه‌های عصبی محسوب می‌شود اما مشکل رایج این تابع، مشکل محو گرادیان (Vanishing Gradient) است. این موضوع به این دلیل اتفاق می‌افتد که برای مقادیر ورودی بزرگ مثبت یا منفی، مشتق تابع Sigmoid، بسیار کوچک (نزدیک به صفر) می‌شود. مشکل محو گرادیان، بخصوص در طول آموزش شبکه‌های عمیق با لایه‌های زیاد با استفاده از پس‌انتشار رخ می‌دهد.



برای مقادیر ورودی مثبت یا منفی بسیار بزرگ، تابع Sigmoid اشباع^۲ می‌شود، یعنی خروجی آن به ۰ یا ۱ بسیار نزدیک می‌شود این به این دلیل است که تابع Sigmoid دارای محدوده‌ای بین ۰ و ۱ بوده و منحنی آن در انتهای صاف می‌شود. هنگامی که خروجی Sigmoid اشباع می‌شود، مشتق آن (شیب)، برای آن مقادیر ورودی بسیار کوچک (نزدیک به صفر) می‌شود. در طول پس‌انتشار،

شیب‌های کوچک از طریق شبکه به عقب منتشر شده و زمانی که در لایه‌های زیادی ضرب می‌شوند، گرادیان‌ها می‌توانند آنقدر کوچک شوند که عملاً محو گردند.



وقتی تابع Sigmoid در هر دو انتهای، اشباع می‌شود (نزدیک به ۰ یا ۱)، گرادیان در این مناطق بسیار نزدیک به صفر است. این بدان معناست که برای مقادیر ورودی بسیار مشبت، خروجی نزدیک به ۱ و برای مقادیر ورودی بسیار منفی، خروجی نزدیک به ۰ است. در هر دو مورد، مشتق تابع Sigmoid بسیار کوچک می‌شود ، که منجر به حداقل تغییرات در وزن‌ها در مرحله به روز رسانی در تمرین می‌شود . از مشتق برای محاسبه گرادیان‌ها در طول پس‌انتشار استفاده شده و تعیین می‌شود وزن‌ها چقدر باید به روز شوند. اگر گرادیان‌ها بسیار کوچک (نزدیک به صفر) باشند، به روز رسانی وزن در طول پس‌انتشار نیز بسیار کم بوده و به همین دلیل از یادگیری موثر یا همگرایی شبکه عصبی به یک راه حل بهینه جلوگیری می‌کند.

مشکل محو گرادیان، روند یادگیری را به ویژه برای نورون‌ها در لایه‌های اولیه شبکه‌های عمیق، به طور قابل توجهی کند می‌کند چون گرادیان به طور تصاعدی در بین لایه‌ها به دلیل قانون زنجیره‌ای که در پس‌انتشار استفاده می‌شود کاهش پیدا می‌کند. یعنی لایه‌های قبلی شبکه، بطور کافی سیگنال‌های خطرا را از لایه‌های بعدی دریافت نکرده و از یادگیری و استخراج ویژگی‌های معنی‌دار

از داده‌های ورودی جلوگیری می‌شود . در نتیجه، شبکه عصبی ممکن است یا همگرا نشد، یا به راه حل غیربهینه همگرا شود و یا این که زمان زیادی طول بکشد تا آموزش داده شود که همگی منجر به عملکرد ضعیف مدل خواهد شد.

چه تابع فعال‌سازی و به چه صورت می‌تواند مشکل را برطرف کند؟

یکی از توابع فعال‌سازی که می‌توان برای برطرف نمودن این مشکل اتخاذ کرد، تابع فعال‌سازی (Rectified Linear Unit) ReLU می‌باشد زیرا در مقادیر مثبت اشباع نشده و همچنین، محاسبه آن بسیار سریع است. این تابع فعال‌سازی بصورت زیر محاسبه می‌شود:

$$f(x) = \max(0, x)$$



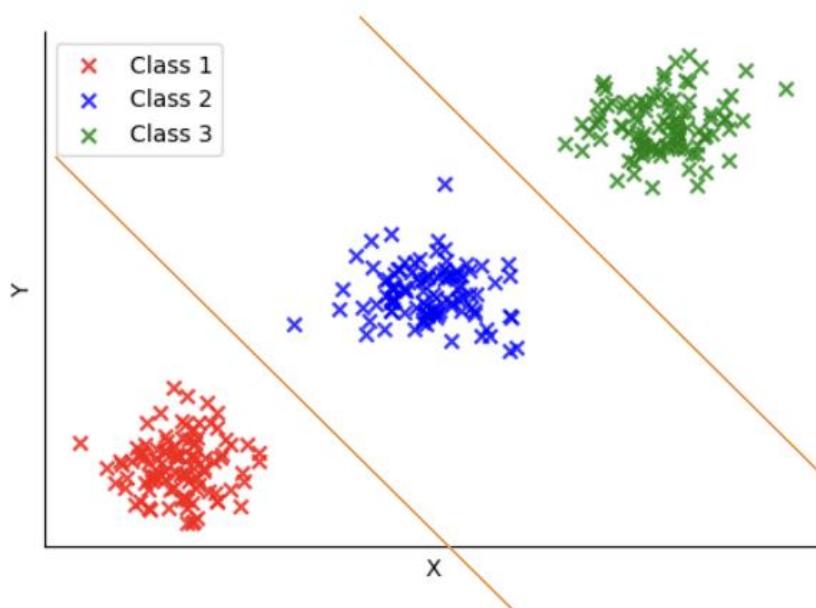
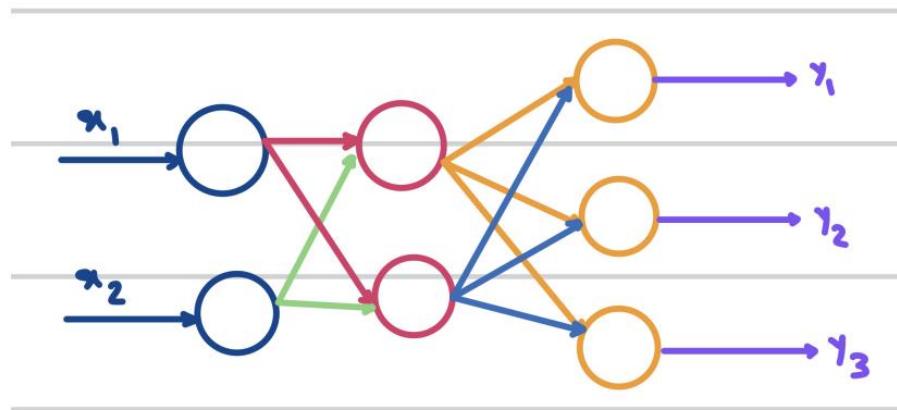
نمودار تابع فعال‌ساز ReLU

ویژگی کلیدی ReLU که به کاهش مشکل محو گرادیان کمک می‌کند، مشتق آن است. مقدار مشتق جزئی loss function، برای ورودی‌های مثبت، دارای گرادیان با مقدار ۱ است، به این معنی که در طول پس‌انتشار، شبکه‌ها با عبور از لایه‌های متوالی کاهش نمی‌یابند. درنتیجه شبکه‌های عمیق می‌توانند به طور مؤثر یاد بگیرند زیرا سیگنال خطا با انتشار مجدد در شبکه از بین نمی‌رود.

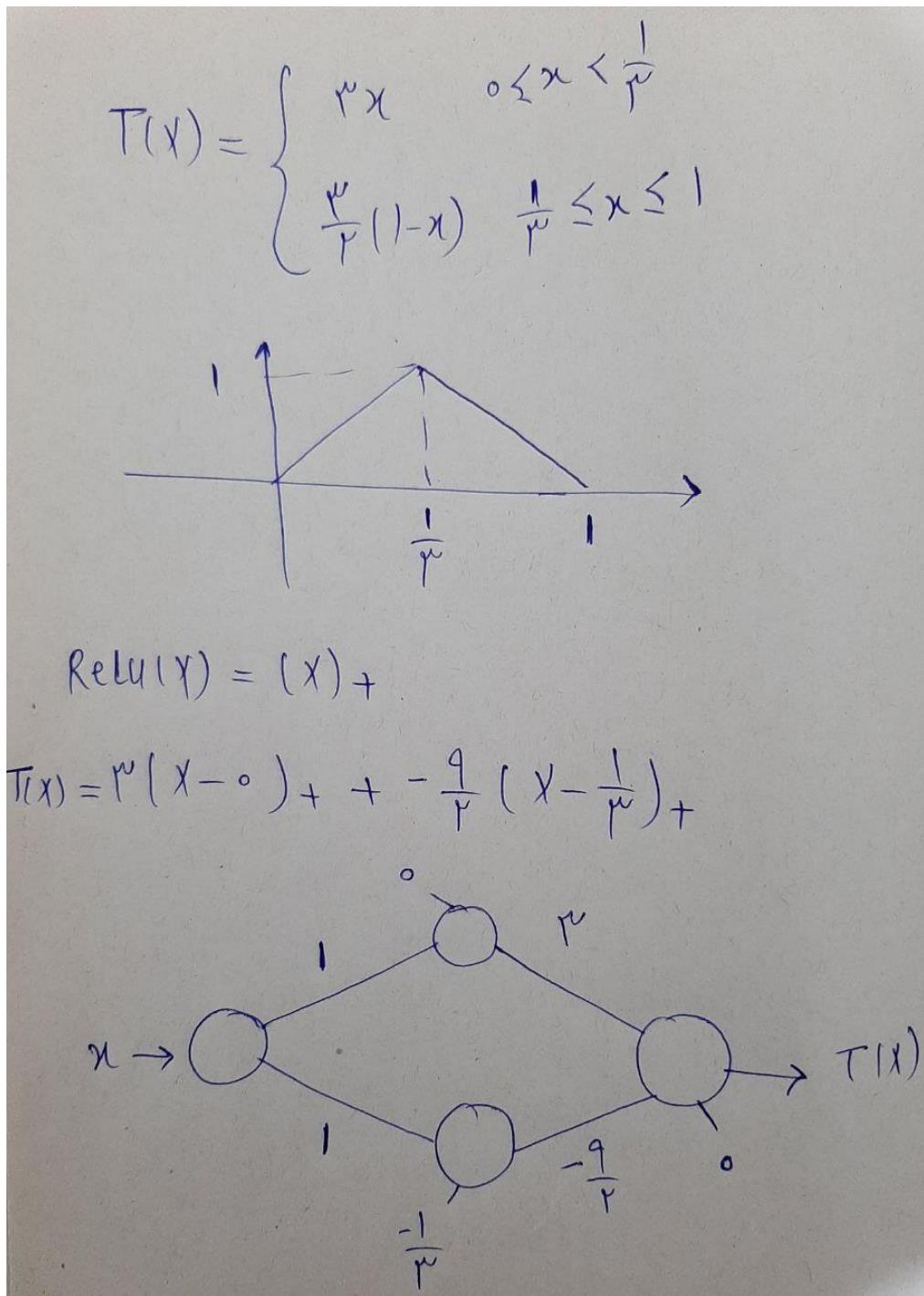
اشارة به دیگر تابع فعال‌سازی با ذکر دلیل که مانند ReLU و Leaky ReLU و PReLU و SELU و ... که مشکل vanishing gradient ندارند نیز قابل قبول است.

(ج)

ج) چون دو ویژگی داریم شبکه دو ورودی خواهد داشت. چون سه کلاس داریم شبکه سه کلاس دارد. داده‌ها با دو خط از هم جدا می‌شوند و یک لایه میانی کافی است. در لایه خروجیتابع فعال‌ساز softmax قرار می‌دهیم. در نتیجه شبکه به شکل زیر توصیف می‌شود:



(۵)

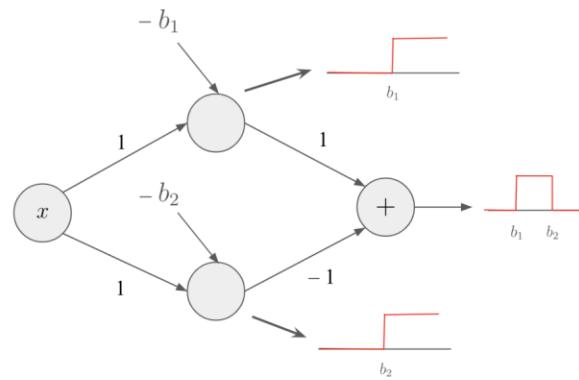


در صورتی که شبکه MLP طراحی شده در بازه $[0,1]$ باشد جواب های دیگر نیز قابل قبول خواهد بود.

- (e) We will first present a one-dimensional example and at the end we will explain how to generalize the approximation to higher dimensions.

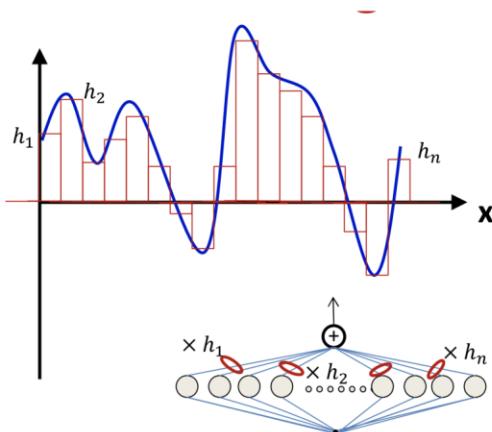
Consider the approximation of some 1-D function such as $\sin(x)$. We show how MLP is a universal approximator and can approximate functions such as $\sin(x)$.

The MLP can approximate these function by creating "pulses" that work similar to bars or rectangles that can approximate even continuous functions as the number of pulses tends to infinite. To create this pulse, we have the following neural network that can be used in larger neural network configurations:

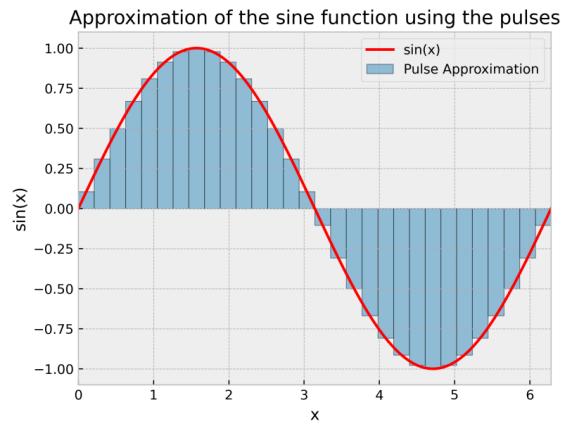


Here, the activation functions in the hidden layer is the step function and the output layer has a linear activation with 0 bias value. In other words, the hidden layer units are Rosenblatt's Perceptron.

Then by summing a large number (a number tending to infinity) this MLP, the final MLP can approximate any function in 1-D.

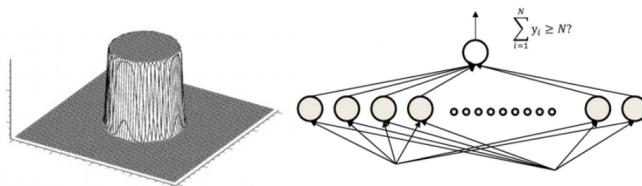


For instance, for $\sin(x)$ this approximation will look like this:

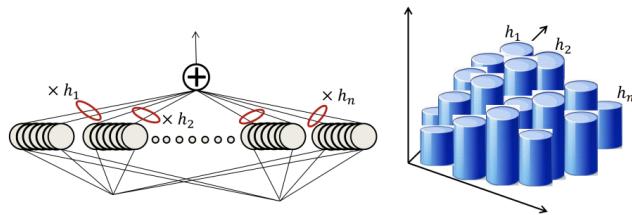


By increasing the number pulses or bars in the plot, i.e. increasing the number of the given MLP configuration, the approximation will converge on the actual function $\sin(x)$. This method of approximating functions using these bars or pulses is similar in spirit to the idea of Riemann sum.

As for the higher dimensions, using an infinite number of Perceptrons in the hidden layer, it is possible to create arbitrary cylinders to then use for the task of approximating in higher dimensions.



By moving and scaling the cylinders and adjusting how thin they are it is possible to approximate the higher dimension functions. Thus, the MLP is a universal approximator.



اگر در جواب خود به Universal approximation theorem با ذکر توضیح و مثال مناسب اشاره کرده باشید، این جواب نیز قابل قبول است.

سؤال ٢

(الف)

We want to calculate the gradient for weight $w_{11}^{(1)}$ and bias $b_1^{(1)}$ with cross-entropy loss for network 5. Note that the activation function for last layer is Softmax and rest has Sigmoid. Now based on this network, lets define some

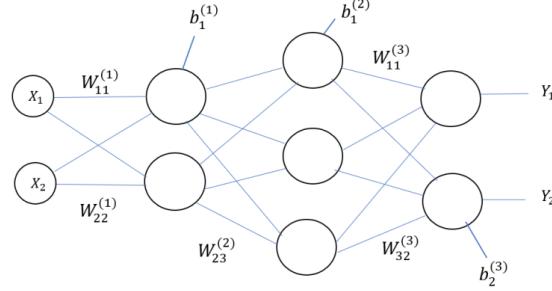


Figure 5: MLP Network

useful variables as follow:

1. Activation Function of Each Neuron: For each neuron i in layer j , we represent it as $z_j^{(i)}$.
2. Each Neuron Input: For each neuron i in layer j , we represent it as $a_i^{(j)}$.
3. Last Layer: For last layer following equation will hold:

$$Y_1 = z_1^{(3)}, \quad Y_2 = z_2^{(3)}$$

As we know the cross-entropy loss is:

$$L_{CE} = -y_1 \log(\hat{y}_1) - y_2 \log(\hat{y}_2) \quad (2)$$

Where y_i is true label and \hat{y}_i is the probability that a sample belongs to class i . Now based on this lets write some equations needed to find the gradient of wanted parameters:

$$\begin{aligned} Y_i &= softmax(a_i^{(3)}) \\ a_i^{(3)} &= \sum_{j=1}^3 W_{ji}^{(3)} z_j^{(2)} + b_i^{(3)} \\ z_i^{(2)} &= sigmoid(a_i^{(2)}) \\ a_i^{(2)} &= \sum_{j=1}^2 W_{ji}^{(2)} z_j^{(1)} + b_i^{(2)} \\ z_i^{(1)} &= sigmoid(a_i^{(1)}) \\ a_i^{(1)} &= \sum_{j=1}^2 W_{ji}^{(1)} X_j + b_i^{(1)} \end{aligned} \quad (3)$$

We know that derivative of sigmoid is:

$$\text{sigmoid}'(x) = \text{sigmoid}(x)(1 - \text{sigmoid}(x)) \quad (4)$$

As we know the softmax equation is:

$$\text{softmax}(a_1^{(3)}) = \frac{e^{a_1^{(3)}}}{e^{a_1^{(3)}} + e^{a_2^{(3)}}} \quad (5)$$

Now, if the last layer has softmax with cross-entropy loss as criterion, assume we want to derivative of Y_1 with respect to $a_1^{(3)}$. (Note that \hat{y}_i is Y_i in this case)

$$\begin{aligned} \frac{\partial L_{CE}}{\partial a_1^{(3)}} &= -y_1 \frac{\partial \log(\hat{y}_1)}{a_1^{(3)}} - y_2 \frac{\partial \log(\hat{y}_2)}{a_1^{(3)}} \\ &= -\frac{y_1}{\hat{y}_1} \frac{\partial \hat{y}_1}{a_1^{(3)}} - \frac{y_2}{\hat{y}_2} \frac{\partial \hat{y}_2}{a_1^{(3)}} \\ &= -y_1 \left(\frac{\frac{e^{a_1^{(3)}}(e^{a_1^{(3)}}+e^{a_2^{(3)}})-e^{2a_1^{(3)}}}{(e^{a_1^{(3)}}+e^{a_2^{(3)}})^2}}{\frac{e^{a_1^{(3)}}}{e^{a_1^{(3)}}+e^{a_2^{(3)}}}} \right) - y_2 \left(\frac{\frac{-(e^{a_1^{(3)}})^2}{(e^{a_1^{(3)}}+e^{a_2^{(3)}})^2}}{\frac{e^{a_1^{(3)}}}{e^{a_1^{(3)}}+e^{a_2^{(3)}}}} \right) \\ &= -y_1 \left(\frac{e^{a_2^{(3)}}}{e^{a_1^{(3)}}+e^{a_2^{(3)}}} \right) + y_2 \left(\frac{e^{a_1^{(3)}}}{e^{a_1^{(3)}}+e^{a_2^{(3)}}} \right) \\ &= -y_1(1 - \text{softmax}(a_1^{(3)})) + y_2(\text{softmax}(a_1^{(3)})) \\ &= -y_1(1 - \hat{y}_1) + y_2(\hat{y}_1) \\ &= -y_1 + \hat{y}_1(y_1 + y_2) \end{aligned} \quad (6)$$

Now we know that either a sample belongs to class 1 or 2. Thus the sum of $y_1 + y_2$ is equal to 1.

$$\frac{\partial L_{CE}}{\partial a_1^{(3)}} = \hat{y}_1 - y_1 \quad (7)$$

Now for finding the gradient of $w_{11}^{(1)}$, we must calculate following derivatives:

$$\begin{aligned} \frac{\partial L_{CE}}{\partial w_{11}^{(1)}} &= \left[\frac{\partial L_{CE}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} + \frac{\partial L_{CE}}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} \right. \\ &\quad + \frac{\partial L_{CE}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_1^{(1)}} + \frac{\partial L_{CE}}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_1^{(1)}} \\ &\quad \left. + \frac{\partial L_{CE}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_3^{(2)}} \frac{\partial z_3^{(2)}}{\partial a_3^{(2)}} \frac{\partial a_3^{(2)}}{\partial z_1^{(1)}} + \frac{\partial L_{CE}}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_3^{(2)}} \frac{\partial z_3^{(2)}}{\partial a_3^{(2)}} \frac{\partial a_3^{(2)}}{\partial z_1^{(1)}} \right] \times \frac{\partial z_1^{(1)}}{\partial a_1^{(1)}} \times \frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}} \end{aligned} \quad (8)$$

Now applying each derivatives for each:

$$\begin{aligned} \frac{\partial L_{CE}}{\partial w_{11}^{(1)}} &= \left[(\hat{y}_1 - y_1)(W_{11}^{(3)})(\sigma(a_1^{(2)}))(1 - \sigma(a_1^{(2)}))(W_{11}^{(2)}) + (\hat{y}_2 - y_2)(W_{21}^{(3)})(\sigma(a_1^{(2)}))(1 - \sigma(a_1^{(2)}))(W_{11}^{(2)}) \right. \\ &\quad + (\hat{y}_1 - y_1)(W_{21}^{(3)})(\sigma(a_2^{(2)}))(1 - \sigma(a_2^{(2)}))(W_{12}^{(2)}) + (\hat{y}_2 - y_2)(W_{22}^{(3)})(\sigma(a_2^{(2)}))(1 - \sigma(a_2^{(2)}))(W_{12}^{(2)}) \\ &\quad \left. + (\hat{y}_2 - y_2)(W_{31}^{(3)})(\sigma(a_3^{(2)}))(1 - \sigma(a_3^{(2)}))(W_{13}^{(2)}) + (\hat{y}_1 - y_1)(W_{23}^{(3)})(\sigma(a_3^{(2)}))(1 - \sigma(a_3^{(2)}))(W_{13}^{(2)}) \right] \\ &\quad \times (\sigma(a_1^{(1)}))(1 - \sigma(a_1^{(1)})) \times X_1 \end{aligned} \quad (9)$$

For bias, the term $\frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}}$ would be deleted:

$$\begin{aligned}\frac{\partial L_{CE}}{\partial w_{11}^{(1)}} &= \left[\frac{\partial L_{CE}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} + \frac{\partial L_{CE}}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} \right. \\ &\quad + \frac{\partial L_{CE}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_1^{(1)}} + \frac{\partial L_{CE}}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial z_1^{(1)}} \\ &\quad \left. + \frac{\partial L_{CE}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_3^{(2)}} \frac{\partial z_3^{(2)}}{\partial a_3^{(2)}} \frac{\partial a_3^{(2)}}{\partial z_1^{(1)}} + \frac{\partial L_{CE}}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_3^{(2)}} \frac{\partial z_3^{(2)}}{\partial a_3^{(2)}} \frac{\partial a_3^{(2)}}{\partial z_1^{(1)}} \right] \times \frac{\partial z_1^{(1)}}{\partial a_1^{(1)}}\end{aligned}\quad (10)$$

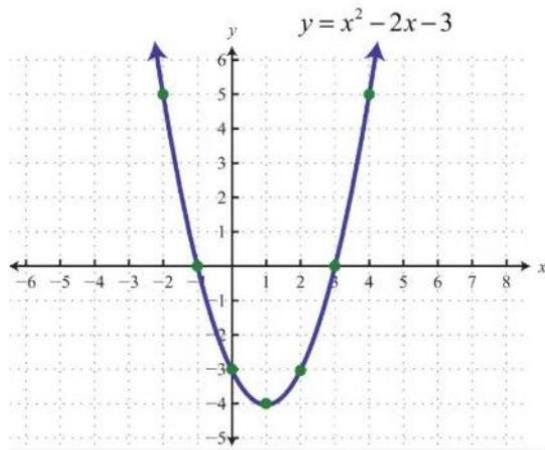
Apply derivatives:

$$\begin{aligned}\frac{\partial L_{CE}}{\partial w_{11}^{(1)}} &= \left[(\hat{y}_1 - y_1)(W_{11}^{(3)})(\sigma(a_1^{(2)}))(1 - \sigma(a_1^{(2)}))(W_{11}^{(2)}) + (\hat{y}_2 - y_2)(W_{21}^{(3)})(\sigma(a_1^{(2)}))(1 - \sigma(a_1^{(2)}))(W_{11}^{(2)}) \right. \\ &\quad + (\hat{y}_1 - y_1)(W_{21}^{(3)})(\sigma(a_2^{(2)}))(1 - \sigma(a_2^{(2)}))(W_{12}^{(2)}) + (\hat{y}_2 - y_2)(W_{22}^{(3)})(\sigma(a_2^{(2)}))(1 - \sigma(a_2^{(2)}))(W_{12}^{(2)}) \\ &\quad + (\hat{y}_2 - y_2)(W_{31}^{(3)})(\sigma(a_3^{(2)}))(1 - \sigma(a_3^{(2)}))(W_{13}^{(2)}) + (\hat{y}_1 - y_1)(W_{23}^{(3)})(\sigma(a_3^{(2)}))(1 - \sigma(a_3^{(2)}))(W_{13}^{(2)}) \\ &\quad \left. \times (\sigma(a_1^{(1)}))(1 - \sigma(a_1^{(1)})) \right]\end{aligned}\quad (11)$$

(ب)

هدف از Gradient Descent چیست؟

گرادیان در اصطلاح ساده به معنای شیب یا شیب سطح است. بنابراین gradient descent در لغت به معنای پایین آمدن از یک شیب برای رسیدن به پایین ترین نقطه در آن سطح است. باید یک نمودار دو بعدی مانند سهمی را در شکل زیر تصور کیم.



در نمودار بالا، پایین ترین نقطه سهمی در $x = 1$ رخ می دهد. هدف الگوریتم gradient descent یافتن مقدار "x" است به طوری که "y" حداقل باشد. "y" در اینجا به عنوانتابع هدفی نامیده می شود که الگوریتم gradient descent روی آن عمل می کند تا به پایین ترین نقطه نزول کند.

من از مسئله رگرسیون خطی برای توضیح الگوریتم نزول گرادیان استفاده می‌کنم. همانطور که از این مقاله به یاد می‌آوریم، هدف رگرسیون به حداقل رساندن مجموع مجذور باقیمانده‌ها (sum of squared residuals) است. می‌دانیم که یک تابع زمانی به حداقل مقدار خود می‌رسد که شیب برابر با 0 باشد. با استفاده از این تکنیک، مسئله رگرسیون خطی را حل کرده و بردار وزن را یاد گرفتیم. همین مشکل را می‌توان با تکنیک نزول گرادیان حل کرد.

نزول گرادیان یک الگوریتم تکراری است که از یک نقطه تصادفی در یک تابع شروع می‌شود و در شیب آن به صورت پله‌ای حرکت می‌کند تا به پایین ترین نقطه آن تابع برسد.

این الگوریتم در مواردی مفید است که نقاط بینه را نمی‌توان با معادل سازی شیب تابع با 0 پیدا کرد. در مورد رگرسیون خطی، می‌توانید مجموع مجذور باقیمانده را به صورت ذهنی به عنوان تابع " y " و بردار وزن را به عنوان " x " در سهمی بالا صورت ذهنی ترسیم کنید.

چگونه در گام‌ها به سمت پایین حرکت کنیم؟

این اصل الگوریتم است. ایده کلی این است که با یک نقطه تصادفی شروع کنیم (در مثال سهمی ما با یک " x " تصادفی شروع کنید) و راهی برای به روز رسانی این نقطه با هر تکرار پیدا کنید به طوری که شیب پایین بیاید.

مراحل الگوریتم عبارتند از

۱. شیب تابع هدف را با توجه به هر پارامتر/ویژگی پیدا کنید. به عبارت دیگر، گرادیان تابع را محاسبه کنید.

۲. یک مقدار اولیه تصادفی برای پارامترها انتخاب کنید. (برای روشن شدن، در مثال سهمی، " y " را با توجه به " x " متمایز کنید. اگر ویژگی‌های بیشتری مانند x^1, x^2 و غیره داشتیم، مشتق جزئی " y " را با توجه به هر یک از ویژگی‌ها می‌گیریم.)

۳. با وصل کردن مقادیر پارامتر، تابع گرادیان را به روز کنید.

۴. اندازه گام‌ها را برای هر ویژگی به صورت زیر محاسبه کنید: اندازه گام = گرادیان * نرخ یادگیری.

$$\text{step size} = \text{gradient} * \text{learning rate}$$

۵. پارامترهای جدید را به صورت:

$$\text{new params} = \text{old params} - \text{step size}$$

پارامترهای جدید = پارامترهای قدیمی - اندازه مرحله محاسبه کنید

۶. مراحل ۳ تا ۵ را تکرار کنید تا گرادیان تقریباً 0 شود.

"نرخ یادگیری" ذکر شده در بالا یک پارامتر انعطاف پذیر است که به شدت بر همگرایی الگوریتم تاثیر می‌گذارد. نرخ یادگیری بزرگتر باعث می‌شود الگوریتم گام‌های بزرگی را به سمت پایین بردارد و ممکن است از نقطه minimum عبور کند و در نتیجه آن را از دست بدهد. بنابراین، همیشه خوب است که به نرخ یادگیری

پایین مانند، پاییند باشد. همچنین می‌توان از نظر ریاضی نشان داد که الگوریتم نزول گرادیان در صورتی که نقطه شروع در بالا باشد، گام‌های بزرگ‌تری به پایین تر از شیب بر می‌دارد و با نزدیک‌تر شدن به مقصد، قدم‌های کوچک‌تری بر می‌دارد تا مراقب باشد که آن را از دست ندهد و همچنین به اندازه کافی سریع باشد.

: Stochastic Gradient Descent (SGD)

چند جنبه منفی در الگوریتم gradient descent وجود دارد. ما باید نگاه دقیق تری به میزان محاسباتی که برای هر تکرار الگوریتم انجام می‌دهیم بیندازیم. فرض کنید 10000 نقطه داده و 10 ویژگی داریم. مجموع مجددور باقیماندها از تعداد نقاط داده تشکیل شده است، بنابراین در مورد ما $10000 = 10 * 1000$ عبارت است. ما باید مشتق اینتابع را با توجه به هر یک از ویژگی‌ها محاسبه کنیم، بنابراین در واقع $10000 = 10 * 1000000$ محاسبه در هر تکرار انجام خواهیم داد. معمول است که 1000 تکرار انجام دهیم، در واقع ما $1000000 = 1000 * 1000000$ محاسبات برای تکمیل الگوریتم داریم. این تقریباً یک سربار است و از این رو نزول گرادیان در داده‌های عظیم کند است.

نزول گرادیان تصادفی (Stochastic gradient descent) به نجات ما می‌آید!! "Stochastic" در اصطلاح ساده به معنای "تصادفی" است.

کجا می‌توانیم به طور بالقوه تصادفی را در الگوریتم نزول گرادیان خود القا کنیم؟ هنگام انتخاب نقاط داده در هر مرحله برای محاسبه مشتقات است. SGD به طور تصادفی یک نقطه داده را از کل مجموعه داده در هر تکرار انتخاب می‌کند تا محاسبات را بسیار کاهش دهد. همچنین معمول است که تعداد کمی از نقاط داده را به جای یک نقطه در هر مرحله نمونه برداری کنید و به آن نزول گرادیان "مینی دسته ای" ("mini-batch" gradient descent) می‌گویند. مینی بچ سعی می‌کند بین خوبی شیب نزول و سرعت SGD تعادل ایجاد کند.

تصادفی گرادیان نزولی (SGD) گونه‌ای از الگوریتم گرادیان نزولی است که برای بهینه سازی مدل‌های یادگیری ماشین استفاده می‌شود. این ناکارآمدی محاسباتی روش‌های سنتی نزول گرادیان را هنگام برخورد با مجموعه داده‌های بزرگ در پروژه‌های یادگیری ماشین بررسی می‌کند.

در SGD، به جای استفاده از کل مجموعه داده برای هر تکرار، تنها یک مثال آموخته تصادفی (یا یک دسته کوچک) برای محاسبه گرادیان و به روز رسانی پارامترهای مدل انتخاب می‌شود. این انتخاب تصادفی، تصادفی بودن را در فرآیند بهینه‌سازی معرفی می‌کند، از این رو اصطلاح "تصادفی" در گرادیان نزولی تصادفی نامیده می‌شود.

مزیت استفاده از SGD کارایی محاسباتی آن است، به ویژه در هنگام برخورد با مجموعه داده‌های بزرگ. با استفاده از یک مثال واحد یا یک دسته کوچک، هزینه محاسباتی در هر تکرار در مقایسه با روش‌های سنتی Gradient Descent که نیاز به پردازش کل مجموعه داده دارند، به طور قابل توجهی کاهش می‌یابد.

الگوریتم نزول گرادیان تصادفی:

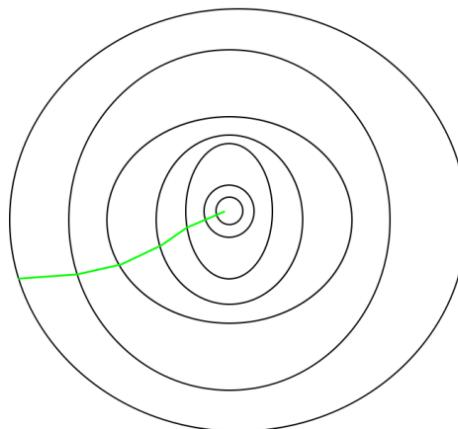
- مقداردهی اولیه: به طور تصادفی پارامترهای مدل را مقداردهی اولیه کنید.
- تنظیم پارامترها: تعداد تکرارها و نرخ یادگیری (آلfa) را برای به روز رسانی پارامترها تعیین کنید.
- حلقه نزول گرادیان تصادفی: مراحل زیر را تکرار کنید تا مدل همگرا شود یا به حداقل تعداد تکرار

بررسد:

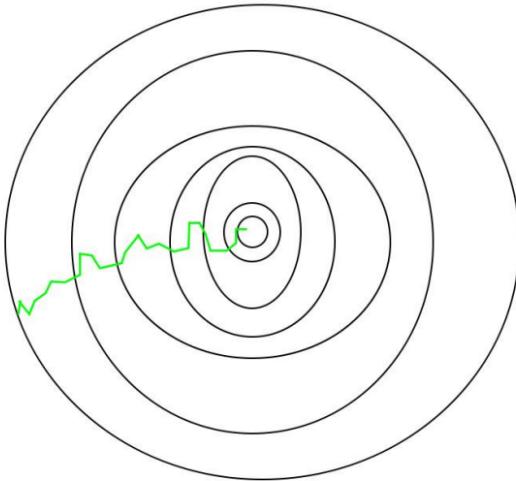
- مجموعه داده آموزشی را به هم بزنید تا تصادفی بودن را معرفی کنید.
- روی هر مثال آموزشی (یا یک دسته کوچک) به ترتیب مخلوط شده تکرار کنید.
- گرادیان تابع هزینه را با توجه به پارامترهای مدل با استفاده از مثال آموزشی فعلی (یا دسته) محاسبه کنید.
- پارامترهای مدل را با برداشتن یک گام در جهت گرادیان منفی که بر اساس نرخ یادگیری مقیاس بندی شده است، به روز کنید.
- معیارهای همگرایی، مانند تفاوت در تابع هزینه بین تکرارهای گرادیان را ارزیابی کنید.
- بازگشت پارامترهای بهینه شده: هنگامی که معیارهای همگرایی برآورده شد یا به حداقل تعداد تکرار رسید، پارامترهای مدل بهینه شده را بروگردانید.

در SGD، از آنجایی که تنها یک نمونه از مجموعه داده به صورت تصادفی برای هر تکرار انتخاب می‌شود، مسیری که الگوریتم برای رسیدن به حداقل‌ها طی می‌کند معمولاً از الگوریتم Gradient Descent معمولی نویزتر است. اما این خیلی مهم نیست زیرا مسیر طی شده توسط الگوریتم مهم نیست، تا زمانی که به حداقل و با زمان آموزش بسیار کوتاه‌تر برسیم.

مسیر طی شده توسط Batch Gradient Descent در زیر نشان داده شده است:



مسیر طی شده توسط Stochastic Gradient Descent به صورت زیر است:



نکته ای که باید به آن توجه کرد این است که، از آنجایی که SGD به طور کلی نویزتر از Gradient Descent معمولی است، به دلیل تصادفی بودن در نزول، معمولاً تعداد تکرارهای بیشتری طول می‌کشد تا به حداقل برسد. حتی با وجود اینکه برای رسیدن به مینیمم نسبت به Gradient Descent معمولی به تعداد تکرارهای بیشتری نیاز دارد، هنوز از نظر محاسباتی بسیار کمتر از Gradient Descent معمولی است. از این رو، در اکثر سناریوهای SGD برای بهینه‌سازی یک الگوریتم یادگیری، نسبت به Batch Gradient Descent ترجیح داده می‌شود.

:Newton-Raphson method

روش نیوتون رافسون یک الگوریتم بهینه سازی شناخته شده است که معمولاً در یادگیری ماشین استفاده می‌شود. این روش می‌تواند برای یافتن minimum یک تابع با پالایش تکراری تخمین اولیه minimum بر اساس گرادیان و مشتق دوم تابع استفاده شود. در یادگیری ماشین، از روش نیوتون رافسون برای یافتن minimum یک تابع هزینه (loss function) استفاده می‌شود که نشان‌دهنده تفاوت بین خروجی پیش‌بینی شده مدل و خروجی هدف واقعی است.

روش نیوتون رافسون به ویژه در یادگیری ماشینی مفید است زیرا در مقایسه با سایر الگوریتم‌های بهینه‌سازی دارای مزایای متعددی است. این شامل:

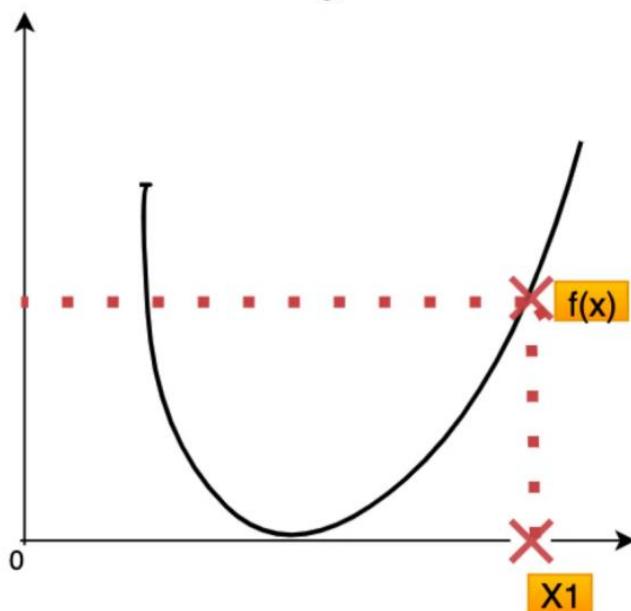
۱. همگرایی سریعتر: روش نیوتون-رافسون معمولاً سریعتر از سایر الگوریتم‌های بهینه سازی، مانند نزول گرادیان، همگرا می‌شود، زیرا انحنای تابع را در نظر می‌گیرد. این به آن اجازه می‌دهد تا با سرعت بیشتری به سمت حداقل حرکت کند.
۲. همگرایی سراسری: برخلاف نزول گرادیان که می‌تواند در حداقل‌های محلی گیر کند، روش نیوتون رافسون تضمین می‌شود که اگر تابع محدب باشد به حداقل جهانی همگرا می‌شود.
۳. استحکام: روش نیوتون رافسون در انتخاب تخمین اولیه قوی است و حساسیت کمتری به انتخاب نرخ

یادگیری دارد.

۴. بهینه‌سازی بهتر توابع پیچیده: روش نیوتن-رافسون می‌تواند توابع پیچیده با minima یا درجه‌های متعدد را به طور مؤثرتری نسبت به سایر الگوریتم‌های بهینه‌سازی مدیریت کند و به عنوان مثال، آن را برای بهینه‌سازی شبکه‌های عصبی عمیق انتخاب بهتری می‌کند.

با این حال، ذکر این نکته ضروری است که روش نیوتن رافسون نیز دارای محدودیت‌هایی است. از نظر محاسباتی گران است، زیرا به محاسبه ماتریس Hessian نیاز دارد که دومین مشتق تابع loss با توجه به پارامترهای مدل است. علاوه بر این، روش می‌تواند به انتخاب تخمین اولیه (initial estimate) حساس باشد، که گاهی اوقات می‌تواند منجر به همگرایی کند یا حتی عدم همگرایی شود.

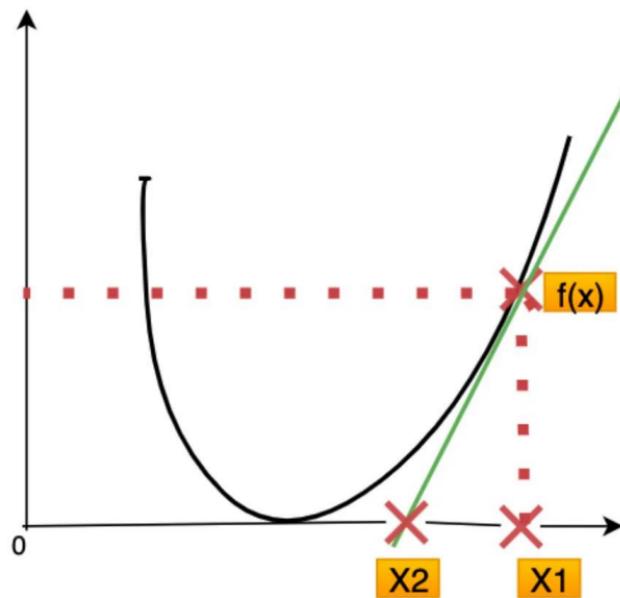
روش نیوتن مبتنی بر مشاهده است که استفاده از مشتق دوم علاوه بر مشتق اول می‌تواند به تقریب بهتر کمک کند. تابع حاصل دیگر خطی نیست بلکه درجه دوم است. برای یافتن ریشه ابتدا با انتخاب یک نقطه تصادفی (X_1) شروع می‌شود و متوجه می‌شوید که تابع در آن مقدار $f(X_1)$ چه چیزی را ارزیابی می‌کند.



حال وقتی $f(x = X_1)$ برابر ارتفاع در نقطه باشد، شیب مشتق این تابع ارزیابی در آن نقطه خواهد بود.

$$\text{شیب } f(x = X_1) =$$

با استفاده از شیب خط می‌توانیم نقطه دوم (X_2) را پیدا کنیم.



نقطه بعدی x_2 است، برای پیدا کردن آن x_1 منهای Δx یا نسبت بین :

$$\frac{f(x_1)}{f'(x_1)}$$

$$\text{Slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x_1)}{\Delta x}$$

$$\Delta x = \frac{f(x_1)}{\text{slope}}$$

$$\text{slope} = f'(x_1)$$

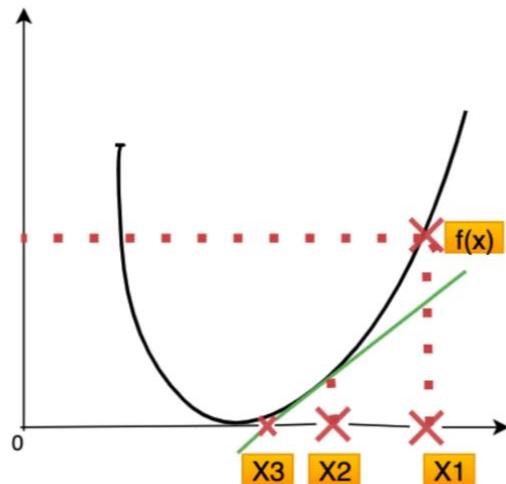
$$\Delta x = \frac{f(x_1)}{f'(x_1)}$$

$$x_2 = x_1 - \Delta x$$

اکنون برای یافتن نقطه بعدی x_3 می توان این فرآیند را تکرار کرد:

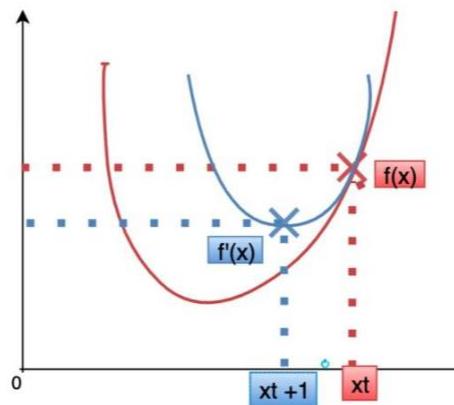
$$\Delta x = \frac{f(x_2)}{f'(x_2)}$$

$$x_3 = x_2 - \Delta x$$



روش نیوتن وقتی برای بهینه‌سازی اعمال می‌شود راهی یافتن حداقل یا حداکثر می‌دهد. این روش مرتبه ۲ را در نظر گرفت زیرا نه تنها از مشتق اول و گرادیان بلکه از مشتق دوم $f''(x)$ که Hassien در چند متغیره نیز نامیده می‌شود استفاده می‌کند.

به جای اینکه مشتق $f(x)$ را بیابد، سعی می‌کند مشتقی را در مشتق تابع ارزیابی شده در X بیابد.



$$x_{t+1} = x_t - \frac{f'(x)}{f''(x)}$$

یک تابع مرتبه اول است که از مشتق آن تابع برای یافتن minimal استفاده می‌کند. روش نیوتن یک الگوریتم ریشه یابی است که از مشتق مرتبه دوم برای یافتن minimal آن تابع استفاده می‌کند. یک مشتق

مرتبه دوم فقط در صورتی می تواند سریعتر باشد که شناخته شده باشد و بتوان آن را به راحتی محاسبه کرد، اما در بیشتر موارد این به محاسبات زیادی نیاز دارد و می تواند گران باشد. اگر برای مشتق اول یک N لازم باشد برای یافتن مشتق دوم یک N^2 لازم است.

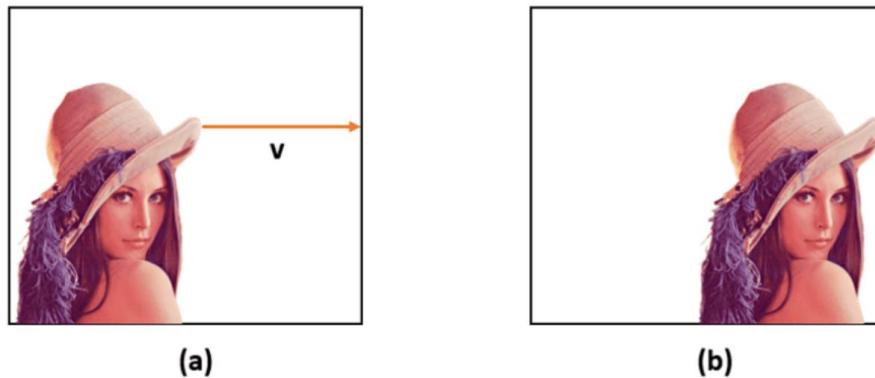
در قسمت مزیایی گفته شده برای الگوریتم بهینه سازی Newton-Raphson در قسمت ۲ در همگرایی سراسری، الگورینم SGD نیز تحت شرایطی خاص در صورت که تابعی که می خواهیم بهینه سازی را انجام دهیم محدب (convex) باشد، اثبات همگرایی به بهینه سراسری دارد.

سوال ۳

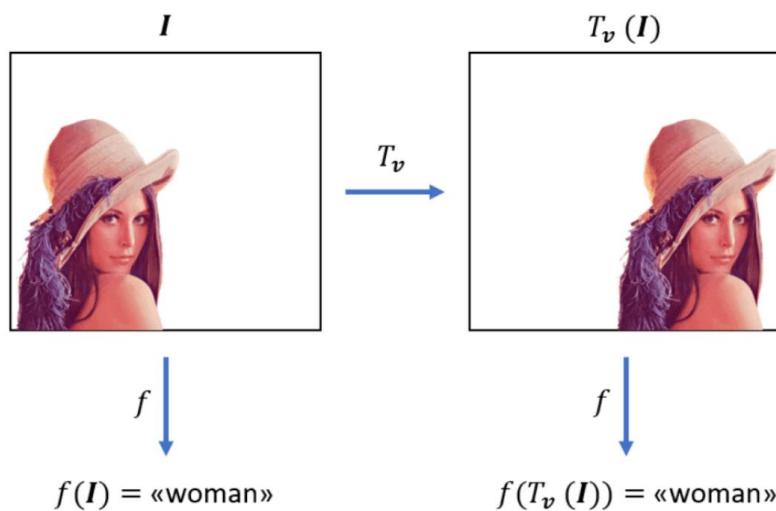
(الف)

یک تبدیل هندسی است که همه نقاط را در یک جهت معین و با فاصله یکسان جابجا می کند. از طرف دیگر، می توان آن را به عنوان لغش مبدأ سیستم مختصات به همان مقدار اما در جهت مخالف تفسیر کرد. در مثال زیر، تصویر (b) از تصویر (a) با جابجایی هر پیکسل ۱۵۰ پیکسل به چپ به دست آمد:

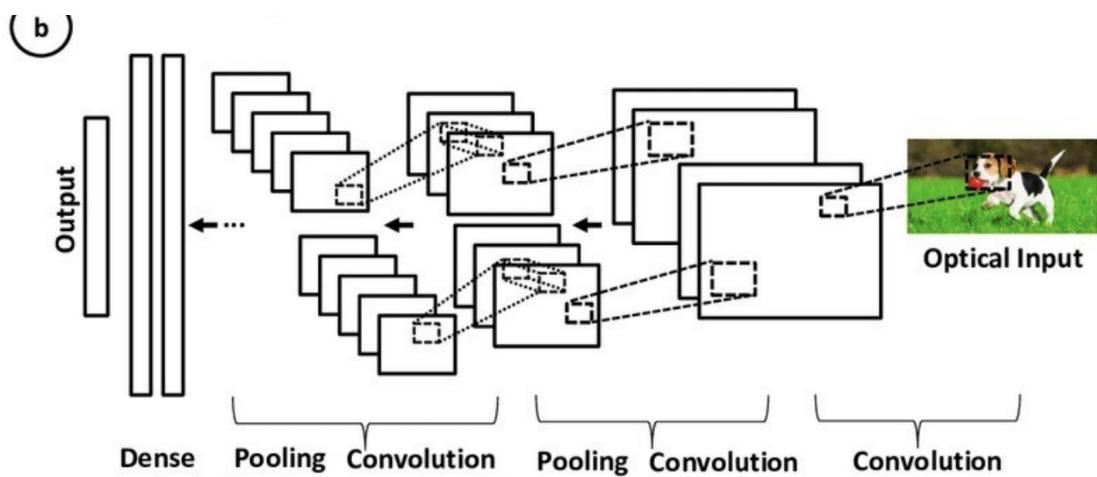
$$T_v(\mathbf{x}) = \mathbf{x} + \mathbf{v}$$



یک ویژگی خاص اگر تحت هیچ *translation* *invariant* تغییر نکند، بیاید تصاویر بالا در نظر بگیریم. ما می توانیم یک زن را در هر دو تصویر تشخیص دهیم حتی اگر مقادیر پیکسل تغییر کرده باشد. یک *image classifier* باید برحسب "زن" را برای هر دو تصویر پیش بینی کند. در واقع خروجی *image classifier* نباید تحت تأثیر موقعیت هدف قرار گیرد. از این رو، خروجی تابع *translation invariant classifier* است.



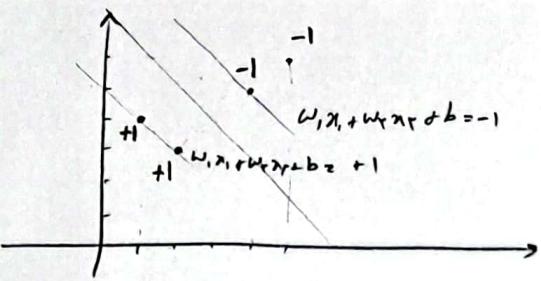
ب



CNN های طراحی شده برای *translation invariant image classification* هستند زیرا اگر ورودی را *translate* کیم، برچسب خروجی تحت تأثیر قرار نخواهد گرفت. به عنوان مثال اگر تصویر ورودی با مقدار مشخصی به سمت راست *translate* شود، *feature map* های ایجاد شده توسط لایه های کانولوشن به همان مقدار و جهت جابه جا می شوند. از طرف دیگر *CNN* ها با استفاده از لایه های ادغام(*pooling*) نیز به دست می آید. عملیات *pooling* معمولاً بر روی نقشه ویژگی(*feature map*) ایجاد شده توسط لایه های کانولوشنی قبلی و توابع فعال سازی غیر خطی اعمال می شود. *pooling* جایگزینی ویژگی ها در یک همسایگی با آماره نشان دهنده، مانند *mean* یا *max* می باشد. از این رو، مکان ویژگی اصلی نادیده گرفته می شود. در نتیجه *translation invariant* در

CNN ها به دلیل عملیات کانولوشن و *pooling* تضمین می کند که ویژگی ها بدون توجه به موقعیت آنها از طریق اشتراک وزن قابل شناسایی هستند، در حالی که *pooling* با خلاصه کردن ویژگی ها در مناطق، استحکام را برای *translation* های کوچک فراهم می کند. با هم، این عملیات به *CNN* ها اجازه می دهد تا الگوها و اشیاء را در موقعیت های فضایی مختلف در تصویر ورودی تشخیص دهند.

کاهش بعد در اندازه **feature map** ها در طی معماری شبکه های *CNN* با استفاده از لایه **pooling** یا خود لایه کانولوشنی (مثلاً مقدار **stride** ۱ بیشتر ۱ قرار دهیم) تاثیر اصلی در اینکه شبکه های *CNN*، ویژگی **Translational invariant** را داشته باشند، دارد.

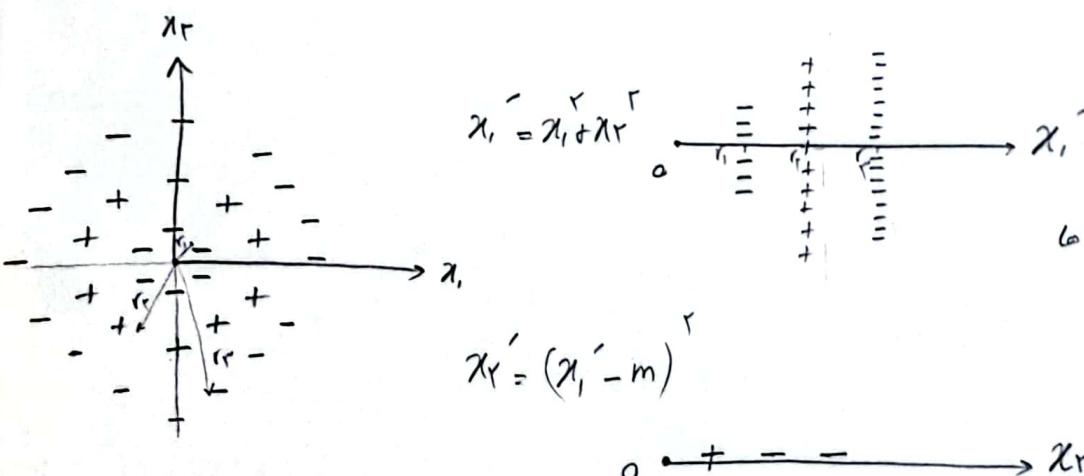
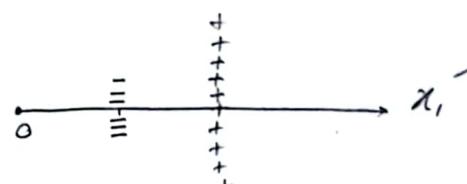
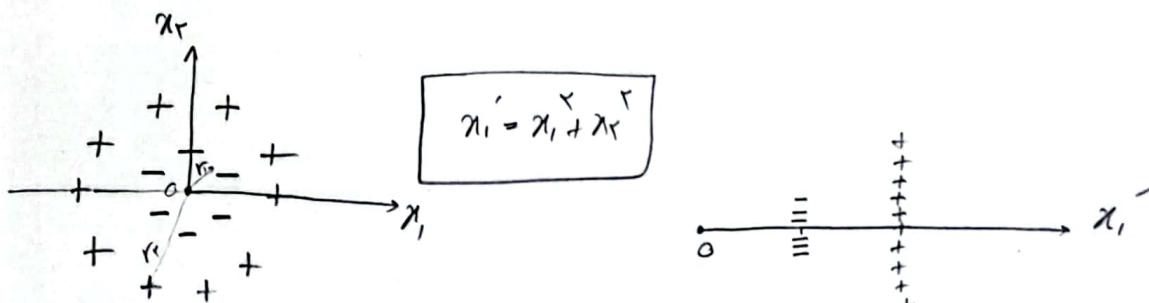
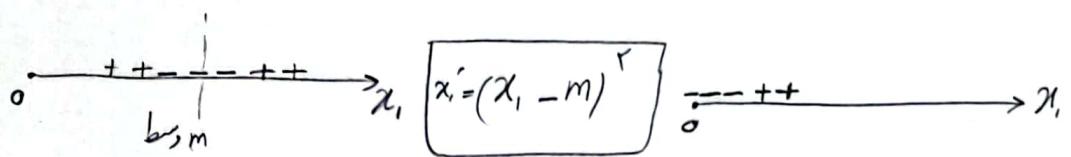


(\leftarrow) - F

$$\begin{cases} w_1 + \gamma w_r + b = -1 \\ w_1 + \gamma w_r + b = +1 \\ \gamma w_1 + \gamma w_r + b = +1 \end{cases} \Rightarrow \begin{aligned} w_1 &= w_r = -\frac{1}{\gamma} \\ b &= \frac{V}{\gamma} \end{aligned} \Rightarrow w^T x + b = .$$

$$\Rightarrow \boxed{-\frac{1}{\gamma} x_1 - \frac{1}{\gamma} x_r + \frac{V}{\gamma} = .}$$

(\leftarrow) - E



$$\text{Let } b \text{ be } \frac{r_1 + r_r + r_m}{\gamma} = m$$

$\therefore x_r' = \left(x_1^\top + x_r^\top - \frac{r_1 + r_r + r_m}{\gamma} \right)^\top$

سیال - کردن احتیاج داشت که فکر کرد آنها تبدیل یافته
ترکیب داده شده باشد (ازم صادر). کردن بعدها در تحریر نمود.

$$K(x, y) = \varphi(x)^T \varphi(y) = \langle \varphi(x), \varphi(y) \rangle$$

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos\alpha$$

$$\rightarrow |\vec{a} \cdot \vec{b}| = \|\vec{a}\| \|\vec{b}\| |\cos\alpha| \rightarrow |\vec{a} \cdot \vec{b}| \leq \|\vec{a}\| \|\vec{b}\|$$

$$\begin{aligned} \xrightarrow{\text{که}} |\vec{a} \cdot \vec{b}| &\leq \|\vec{a}\| \|\vec{b}\| \\ \vec{a} = \varphi(x) & \\ \vec{b} = \varphi(y) & \end{aligned} \quad \left. \begin{aligned} \rightarrow |\varphi(x)^T \varphi(y)| &\leq \underbrace{\|\varphi(x)\|}_{\langle \varphi(x), \varphi(x) \rangle} \underbrace{\|\varphi(y)\|}_{\langle \varphi(y), \varphi(y) \rangle} \\ & \end{aligned} \right\}$$

$$\rightarrow \boxed{k(x, y) \leq k(x, x) k(y, y)}$$

$$\left. \begin{aligned} \mu_\varphi &= \frac{1}{Q} \sum_{n=1}^Q \varphi(x_n) \\ \mu_\varphi^T &= \frac{1}{Q} \sum_{m=1}^Q \varphi(x_m)^T \end{aligned} \right\} \xrightarrow{\text{که}} \mu_\varphi^T \mu_\varphi = \frac{1}{Q} \left(\sum_{m=1}^Q \varphi(x_m)^T \right) \left(\sum_{n=1}^Q \varphi(x_n) \right) = \frac{1}{Q} \sum_{m=1}^Q \sum_{n=1}^Q \underbrace{\varphi(x_m)^T \varphi(x_n)}_{k_\varphi(x_m, x_n)}$$

$$\Rightarrow \|\mu_\varphi\|^r = \frac{1}{Q} \sum_{m=1}^Q \sum_{n=1}^Q k_\varphi(x_m, x_n)$$

$$\boxed{\|\mu_\varphi\| = \sqrt{\frac{1}{Q} \sum_{m=1}^Q \sum_{n=1}^Q k_\varphi(x_m, x_n)}}$$