به‌نام خدا

تئوری یادگیری ماشین

دکتر سیدصالحی

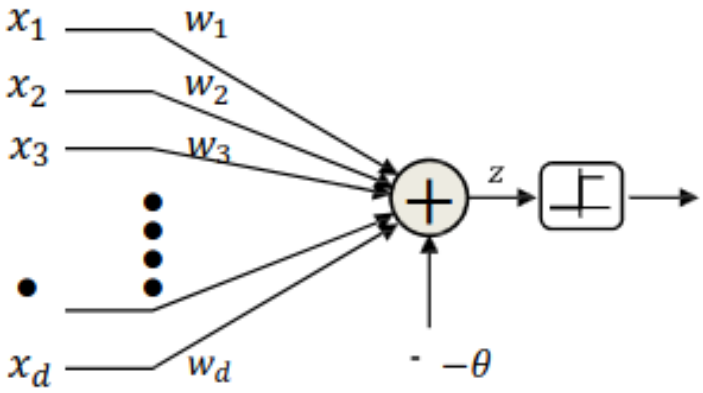
جلسه پانزدهم

دانشکده ریاضی و علوم کامپیوتر

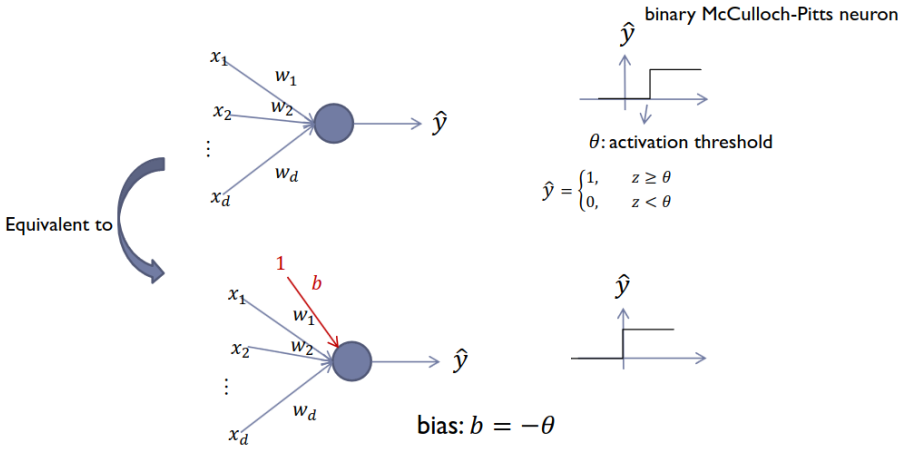
اردیبهشت ماه ۱۴۰۳

Neural Networks

مدل های اولیه ریاضیاتی که شکل گرفتند بر اساس عملکرد سلول های عصبی مغز انسان بودند.



پس از این مدل های اولیه طراحی دیگری پیش آمد که آستانه ای که میخواهند در نورون طراحی شده چک کنند را به شکل بایاس به نورون پاس بدهند و علامت نتیجه را به عنوان خروجی مطرح کنند.



این تغییر ما را به سمت طراحی مدل $perceptron$ میبرد. حال که مدل و فرم کلی این روش را میدانیم به سراغ الگوریتم یادگیری آن میرویم.

learning algorithms

perceptron

- Initialize $\boldsymbol{w}$
- Cycle through the training instance
- While more classification errors
  - For $i = 1 \dots N_{train}$

$$\hat{y}^{(i)} = sign(\boldsymbol{w}^T \boldsymbol{x}^{(i)})$$

    - If $\hat{y}^{(i)} \neq y^{(i)}$

$$\boldsymbol{w} = \boldsymbol{w} + y^{(i)} \boldsymbol{x}^{(i)}$$

update rule

Perceptron: If $\text{sign}(\boldsymbol{w}^T \boldsymbol{x}^{(n)}) \neq y^{(n)}$ then

$$\nabla J_n(\boldsymbol{w}^t) = -\eta \boldsymbol{x}^{(n)} y^{(n)}$$

$$J_n(\boldsymbol{w}) = -\boldsymbol{w}^T \boldsymbol{x}^{(n)} y^{(n)}$$
if misclassified

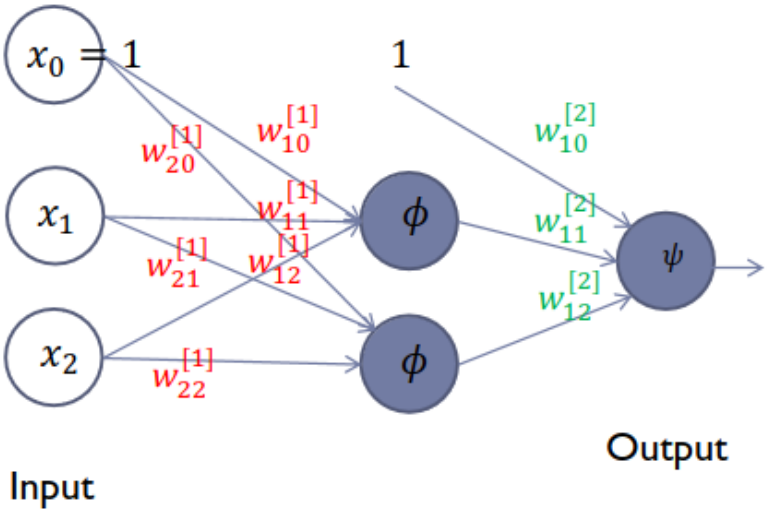
ADALINE

update rule

- ADALINE: $\nabla J_n(\boldsymbol{w}^t) = -\eta(y^{(n)} - \boldsymbol{w}^T \boldsymbol{x}^{(n)}) \boldsymbol{x}^{(n)}$
  - Widrow-Hoff, LMS, or delta rule

$$J_n(\boldsymbol{w}) = \left(y^{(n)} - \boldsymbol{w}^T \boldsymbol{x}^{(n)}\right)^2$$

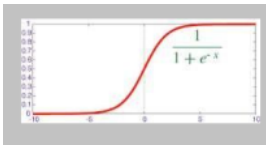
یکی از راهکار های افزایش دقت مدل ها گسترش مدل ها با استفاده از اضافه کردن لایه های پنهان است. اینجا با مفهوم *feed forward* آشنا میشویم که دیتا ورودی را میگیرد و از لایه اول شروع میکند و همینطور لایه به لایه محاسبات را انجام میدهد و به خروجی ختم میشود. یکی از مدل های معروف این دسته از مدل ها *multi layer perceptron* است.



یکی از ویژگی های خوبی که ما را تشویق به استفاده از شبکه های عصبی میکند این است که هر لایه نمایانگر دسته ای از فیچر ها است و میتواند معنی دار باشد و ما از این اطلاعات بدست آمده در طول یادگیری میتوانیم استفاده کنیم.

Sigmoid

عامل غیرخطی کننده ای که در مدل بالا میتوانید مشاهده کنید *activation function* نام دارند که یکی از پرکاربردترین آنها تابع *sigmoid* است.



$$f(z) = \frac{1}{1 + \exp(-z)} \qquad f'(z) = f(z)(1 - f(z))$$

BackPropagation

بیاید فرض کنیم که از روش *gradient descent* برای اپدیت کردن پارامترها استفاده میکنیم. در این روش همانطور که به خاطر دارید به گرادیان نیاز داریم. اما پیچیدگی که در شبکه های عصبی داریم اینجاست که توابعی که در خروجی از نورون استفاده میکنیم در بحث مشتق پذیری دچار مشکل هستند و دیگر مساله بهینه کردن محسبات برای حساب گرادیان است.

- $\boldsymbol{h}_{\boldsymbol{W}}^{(n)}$ as the output of a network with $\boldsymbol{n}$ layers

$$J = \sum_{n=1}^{N} loss\left(\boldsymbol{h}_{\boldsymbol{W}}^{(n)}, \boldsymbol{y}^{(n)}\right)$$

و بنابر الگوریتم داریم

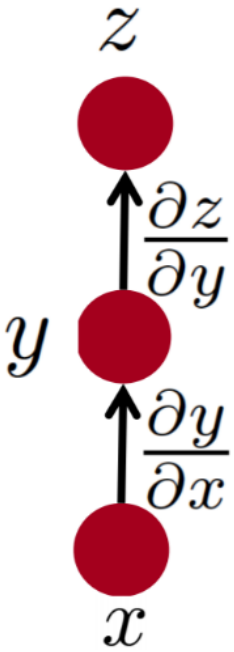
For every layer k for all i, j update:

$$w_{ji}^{[k]} = w_{ji}^{[k]} - \eta \frac{dJ}{dw_{ji}^{[k]}}$$

تا هنگامی که J همگرا شود.

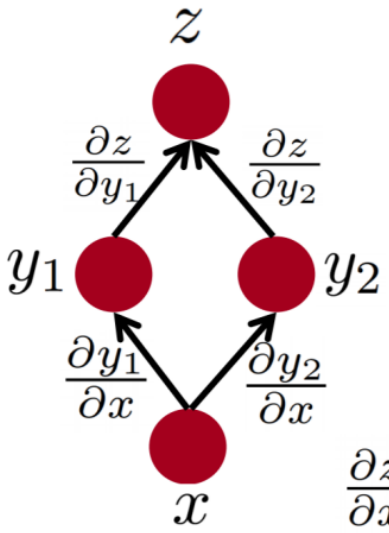
Chain Rule

قانونی است که در فرایند مشتق گیری و محاسبه گرادیان مورد استفاده قرار میگیرد.



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

اتفاقی که وقتی میخواهیم توی شبکه های عصبی مشتق بگیریم در یک نورون می افتد بدین شکل است.



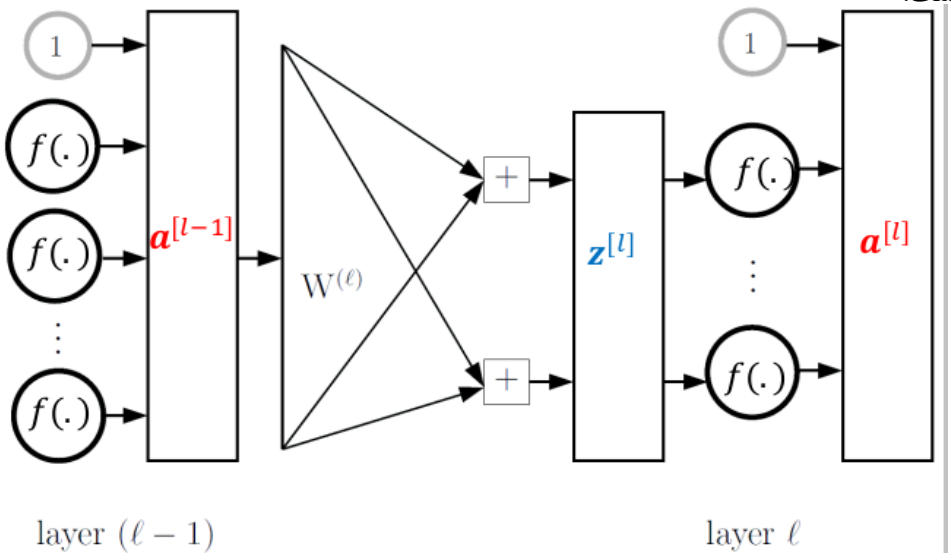
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

چیزی که ما برای الگوریتممان نیاز داریم نحوه بدست آوردن

$$\frac{dJ}{dw_{ij}^{[k]}}$$

است.

حال یک شبکه عصبی را در نظر بگیرید که $l-1$ لایه نهان دارد. ورودی های این شبکه را به شکل $a^{[0]}$ نمایش میدهیم و خروجی آن را به فرمت $a^{[l]}$ معرفی میکنیم. تابع f هم نمایانگر $activation\ function$ ها است.



حال طبق قاعده زنجیره ای داریم.

$$\frac{\partial loss}{\partial w_{ji}^{[l]}} = \frac{\partial loss}{\partial a_j^{[l]}}\frac{\partial a_j^{[l]}}{\partial w_{ji}^{l}}$$

مشتق خروجی نسبت به وزن های لایه اخر چیزی است که به راحتی با استفاده از مشتق $activation function$ و مشتق ورودی این تابع نسبت به وزن های این لایه محاسبه میشود.

$$\frac{\partial a^{[l]}}{\partial w_{ji}^{[l]}} = f'(z_j^{[l]}) \frac{\partial z_j^{[l]}}{\partial w_{ji}^{l}} = f'(z_j^{[l]}) a_i^{l-1}$$

پس در نهایت برای لایه اخر داریم.

$$\frac{\partial loss}{\partial w_{ji}^{l}} = \frac{\partial loss}{\partial a_j^{[l]}} f'(z_j^{[l]}) a_i^{[l-1]}$$

و اما گرادیان گیری در لایه های قبلی آیا به راحتی لایه اخر است؟

$$\frac{\partial loss}{\partial w_{ji}^{[l]}} = \frac{\partial loss}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial w_{ji}^{[l]}} = \frac{\partial loss}{\partial z_j^{[l]}} a_i^{[l-1]}$$

حال چگونه باید برای لایه های درونی این مشتق را بدست آورد؟

$$\frac{\partial loss}{\partial z_i^{[l-1]}}$$

حال اگر از مواردی که بالاتر مطرح شد استفاده کنیم به این نتیجه میرسیم که برای محاسبه این مشتق نیازمند مشتق های لایه های بالاتر بصورت بازگشتی هستیم.

$$\frac{\partial loss}{\partial z_i^{[l-1]}} = \frac{\partial a_i^{[l-1]}}{\partial z_i^{[l-1]}} \Sigma_{j=1}^{d^{[l]}} \frac{\partial loss}{\partial z_j^{[l]}} \times \frac{\partial z_j^{[l]}}{\partial a_i^{[l-1]}}$$

که نتیجتا داریم.

$$= f'(z_i^{[l-1]}) \Sigma_{j=1}^{d^{[l]}} \frac{\partial loss}{\partial z_j^{[l]}} \times w_{ji}^{[l]}$$

حال که این روابط را بدست آوردیم به سراغ مفهوم $backpropagation$ میرویم.

$$\frac{\partial loss}{\partial w_{ji}^{[l]}} = \frac{\partial loss}{\partial z_j^{[l]}} \times \frac{\partial z_j^{[l]}}{\partial w_{ji}^{[l]}}$$

$$= \delta_j^{[l]} \times a_i^{[l-1]}$$

همانطور که در تصویر پایین هم میتوانید مشاهده کنید برای اینکه مشتق نسبت به یکی از وزن ها را بیابیم نیازمند اطلاعاتی از لایه های بالاتر نیاز داریم که سبب میشود تا ما مجبور باشیم برای محاسبه گرادیان یک مرحله $feedforward$ تا انتهای شبکه بریم و سپس از انتها شروع به مشتق گرفتن بکنیم که این عمل تحت عنوان $backpropagation$ شناخته میشود.

$$\delta_j^{[l]} = \frac{\partial loss}{\partial z_j^{[l]}}$$

$$\delta_i^{[l-1]} = f'(z_i^{[l-1]}) \Sigma_{j=1}^{d^{[l]}} \delta_j^{[l]} \times w_{ji}^{[l]}$$