

**گزارشکار سوم آزمایشگاه درس**

**مقدمه ای بر هوش محاسباتی**

استاد احمدی موسوی

دانشکده مهندسی برق

اعضای گروه:

ریحانه آهنی 9823009

مهدیه سادات بنیس 9823045

سمیرا سلجوقی 9823048

پاییز 1401

فهرست مطالب

[**فصل اول، پیش گزارش** 2](#_Toc118731220)

[**1-1 محدودیت های مدل پرسپترون** 2](#_Toc118731221)

[**1-2 ساختار کلی شبکه عصبی دو لایه** 2](#_Toc118731222)

[**1-3 مراحل الگوریتم پس انتشار خطا** 3](#_Toc118731223)

[**فصل دوم، شرح آزمایش** 7](#_Toc118731224)

[**2-1 پیاده سازی شبکه عصبی دو لایه** 7](#_Toc118731225)

[**2-2 تست مدل** 7](#_Toc118731226)

[**فصل سوم، تمرین** 7](#_Toc118731227)

[**3-1 پیاده سازی مدل در محیط متلب** 7](#_Toc118731228)

[**3-2 داده های ایستگاه آب و هوایی** 7](#_Toc118731229)

[**3-2-1 دسته بندی داده ها** 7](#_Toc118731230)

[**3-2-2 شبکه بیش از 3 لایه در متلب** 7](#_Toc118731231)

[**3-2-3بهبود داده ها** 7](#_Toc118731232)

[**3-2-4 بررسی داده ها و over fit** 7](#_Toc118731233)

[**3-2-5 تاثیر افزایش لایه ها** 7](#_Toc118731234)

[**3-2-6 بخش امتیازی** 7](#_Toc118731235)

[**منابع و مراجع** 7](#_Toc118731236)

# **فصل اول، پیش گزارش**

## **1-1 محدودیت های مدل پرسپترون**

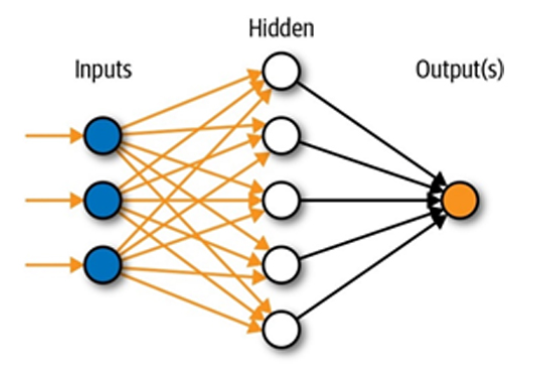
Perceptron یکی از ساده‌ترین معماری‌های ANN است.پرسپترون های تک لایه می توانند فقط الگوهای قابل تفکیک خطی را بیاموزند. مطالعاتی که روی پرسپترون در دهه 60 انجام شد نشان داد که این شبکه‌ها برخی مسائل را به راحتی حل می‌کنند. اما حل یکسری مسائلِ نه چندان سخت با پرسپترون غیر ممکن به نظرمی‌رسد. Marvin Minsky و Seymour Papert در سال 1969 کتابی به نام پرسپترون نوشتند که در آن تمامی توانایی‌ها و محدودیت‌های پرسپترون را مطرح کردند. آن‌ها بیان کردند که پرسپترون تنها قادر به حل مسائلی است که linearly separable باشند، یعنی به شکل خطی تفکیک پذیر باشند. و این شبکه در حل مسائلی مانند XOR ناتوان است.

اگرچه کلاس‌بندهای خطی دیگر مانند Logistic Regression نیز نمی‌توانستند چنین مسئله‌ای را حل کنند! اما انتظاری که محققان از پرسپترون داشتند بسیار بیشتر بود. همین مسئله باعث شد که بسیاری از آن‌ها کاملااز پرسپترون و شبکه‌های عصبی قطع امید کنند. در نتیجه حوزه شبکه‌های عصبی به یک خواب زمستانی عمیق فرو رفت! تا زمانی که الگوریتم پس‌انتشار خطا مطرح شد[1].

مدل در نظر گرفته شده براي پرسپترون از محدوديت هاي زيادي رنج مي برد براي مثال تنها توابع جداپذير خطي امكان دسته بندي توسط اين نوع مدل را دارند. شبكه هاي عصبي چندلايه مدل توسعه يافته پرسپترون هستند و مشكلات مدل پرسپترون را ندارند.

## **1-2 ساختار کلی شبکه عصبی دو لایه**

ساختار کلی یک شبکه عصبی دولایه به صورت زیر است:



اجزای شبکه عصبی شامل موارد زیر می باشد:

1) وزن ها و بایاس(weights and bias): وزن­ها مقادیر عددی هستند که در ورودی­ ها ضرب می­شوند. در انتشار مجدد، آنها برای کاهش ضرر اصلاح می­شوند. به عبارت ساده، وزن­ها مقادیری هستند که از شبکه­ های عصبی توسط ماشین آموخته می­شوند. آنها بسته به تفاوت بین خروجی­ های پیش بینی شده در مقابل ورودی­ های آموزشی، خود تنظیم می­شوند.

2) تابع فعالساز (Activation function): عملکرد فعال سازی یک فرمول ریاضی است که به نورون کمک می­کند تا روشن / خاموش شود.

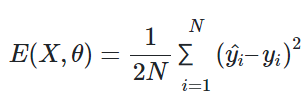
3) لایه ورودی (Input layer): لایه ورودی ابعاد بردار ورودی را نشان می­دهد.

4) لایه پنهان (Hidden layer): لایه پنهان، گره ­های واسطه­ ای است که فضای ورودی را به مناطقی با مرزهای نرم تقسیم می­کنند. مجموعه­ ای از ورودی وزنی را گرفته و از طریق یک تابع فعال سازی خروجی تولید می­کند.

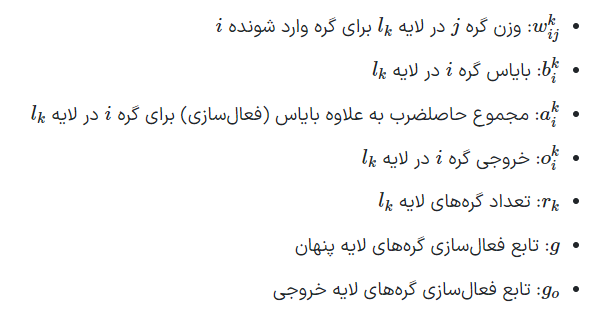
5) لایه خروجی (Output layer): لایه خروجی نشان دهنده خروجی شبکه عصبی است[2] .

## **1-3 مراحل الگوریتم پس انتشار خطا**

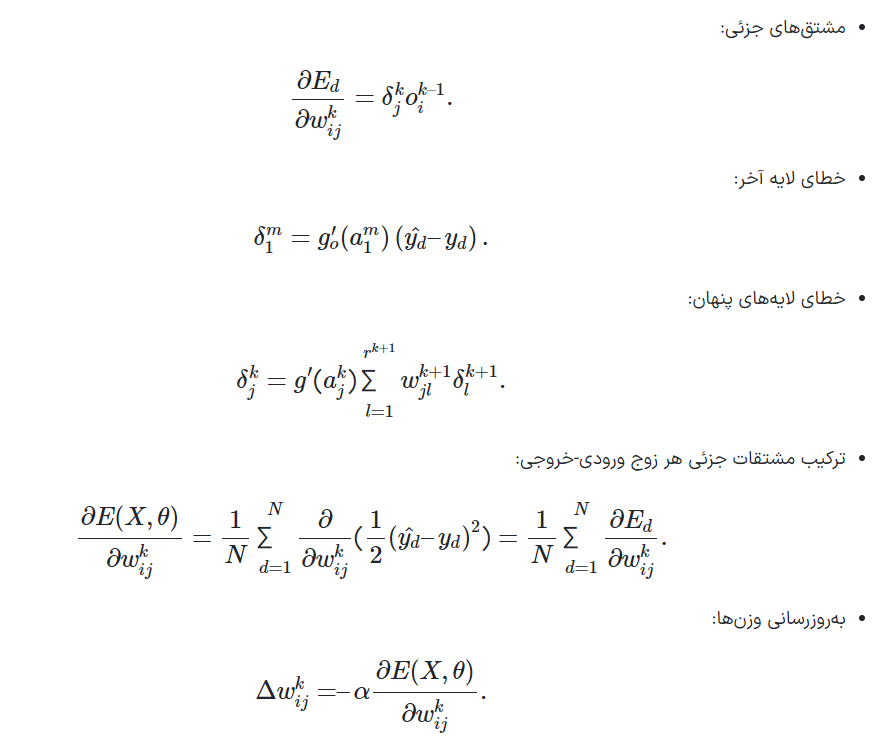
فرمول‌بندی زیر برای شبکه عصبی با یک خروجی است، اما این الگوریتم با استفاده مداوم از قاعده [مشتق زنجیره‌ای](https://blog.faradars.org/chain-rule/) و قانون توان قابلیت پیاده‌سازی روی یک شبکه با هر تعداد خروجی را دارد. بنابراین، برای همه مثال‌های زیر، زوج ورودی-خروجی به فرم  (x,y) است. تابع خطا در پس انتشار کلاسیک، خطای میانگین مربعات است:



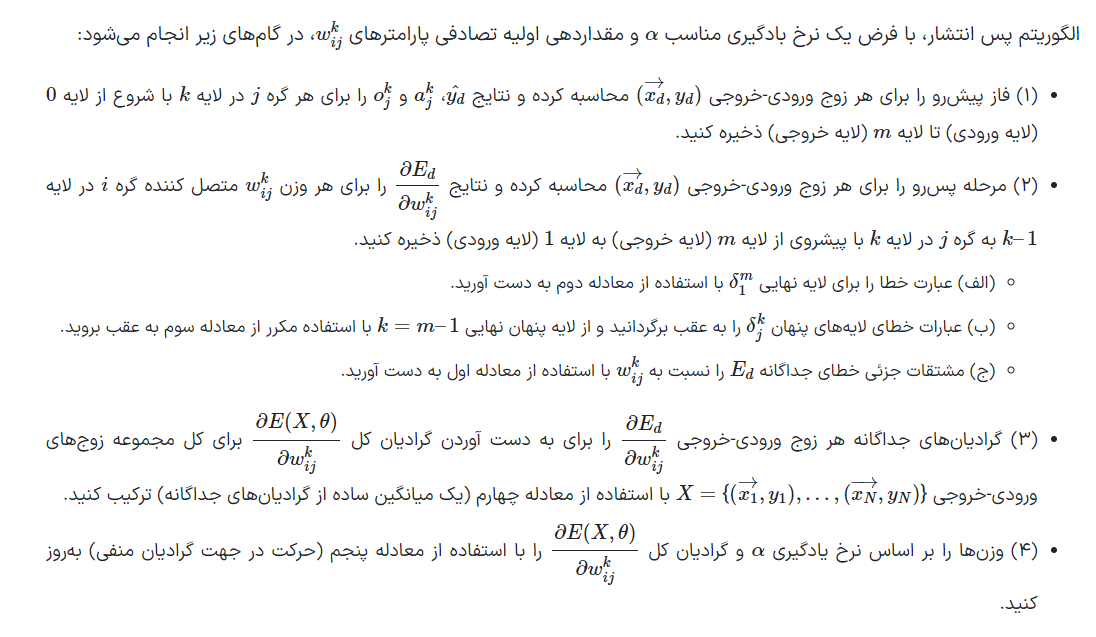
پارامتر های استفاده شده به صورت زیر هستند:



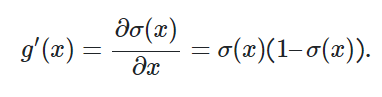
الگوریتم پس انتشار به پنج معادله زیر بستگی خواهد داشت:



الگوریتم پس انتشار، با فرض یک نرخ بادگیری مناسب  α و مقداردهی اولیه تصادفی پارامترهای  ، در گام‌های زیر انجام می‌شود:

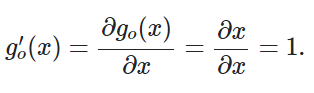


الگوریتم کلاسیک پس انتشار برای مسائل [رگرسیون](https://blog.faradars.org/simple-linear-regression/) با واحدهای فعال‌سازی سیگموئید طراحی شده است. در حالی که پس انتشار می‌تواند برای مسائل طبقه‌بندی و همچنین شبکه‌هایی با توابع فعال‌سازی غیرسیگموئیدی کاربرد داشته باشد، تابع سیگموئید خواص ریاضی مناسبی دارد که وقتی با یک تابع فعال‌سازی خروجی مناسب ترکیب شود، درک الگوریتم را بسیار ساده می‌کند. بنابراین، در فرمول کلاسیک، تابع فعال‌سازی برای گره‌های پنهان سیگموئید (g(x)=σ(x)) و تابع فعال‌سازی خروجی، [تابع همانی](https://blog.faradars.org/%D8%AA%D8%A7%D8%A8%D8%B9-%D9%87%D9%85%D8%A7%D9%86%DB%8C-%D9%88-%D8%AE%D8%B5%D9%88%D8%B5%DB%8C%D8%A7%D8%AA-%D8%A2%D9%86/) (go(x)=x) است. خروجی شبکه فقط مقدار وزن‌داری از لایه پنهان آن، یعنی فعال‌سازی است. پس انتشار در واقع یک عامل اصلی محرک در استفاده تاریخی از توابع فعال‌سازی سیگموئید به دلیل مشتق مناسب آن است:



بنابراین، محاسبه مشتق تابع سیگموئید چیزی بیش از به خاطر سپردن خروجی σ(x)  و قرار دادن آن در معادله بالا نیاز ندارد.

علاوه بر این، مشتق تابع فعال‌سازی خروجی نیز بسیار ساده است:



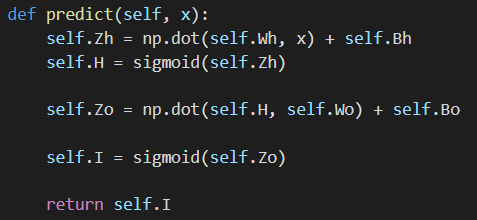
بنابراین، با استفاده از این دو تابع فعال‌سازی، نیاز به یادآوری مقادیر فعال‌سازی را حذف می‌کند و باعث کاهش چشمگیر اثر حافظه الگوریتم می‌شود. این امر به این دلیل است که مشتق برای تابع فعال‌سازی سیگموئید در مرحله پس‌رو فقط باید خروجی آن تابع را در مرحله پیش‌رو به یاد بیاورد و به مقدار فعال‌سازی واقعی وابسته نیست. به طور مشابه، مشتق برای تابع فعال‌سازی همانی به چیزی وابسته نیست، زیرا یک عدد ثابت است[3] .

# **فصل دوم، شرح آزمایش**

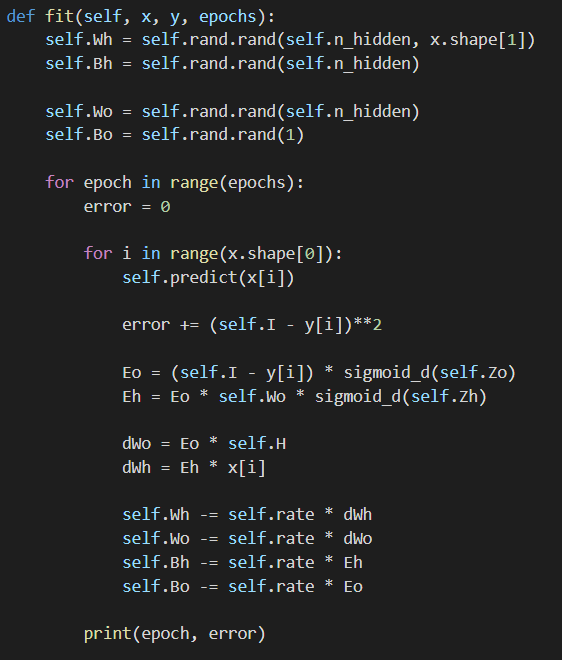
## **2-1 پیاده سازی شبکه عصبی دو لایه**

در ابتدا برای ایجاد شبکه عصبی دولایه کلاسی به نام DNN (Deep Neural Network) ایجاد میکنیم. این کلاس ورودی های زیر را دریافت می کند:

1. تعداد نورون ها در لایه پنهان
2. نرخ یادگیری ( به طور پیش فرض 0.5 )
3. و seed تعیین کننده اعداد تصادفی برای مقداردهی اولیه وزن ها

سپس تابعی به نام predict برای محاسبه خروجی شبکه می سازیم. در این تابع ابتدا x ورودی با وزن های لایه پنهان ضرب نقطه ای میشود و سپس bias لایه پنهان به آن اضافه می شود. این مقدار به نام Zh در کلاس ذخیره میشود. به سپس از تابع sigmoid عبور کرده و خروجی لایه پنهان می باشد و در عضوی در کلاس به اسم H ذخیره میشود.

سپس مقدار عضو H کلاس با وزن های خروجی ضرب نقطه ای شده و bias لایه خروجی به آن اضافه میشود. این مقدار با نام Zo در کلاس ذخیره میشود. در نهایت خروجی لایه خروجی با عبور این مقدار از تابع sigmoid محاسبه شده و در عضو I در کلاس ذخیره میشود. همچنین مقدار I به عنوان خروجی تابع بازگردادنده میشود.

برای تازه سازی وزن های شبکه، تابعی به نام fit تعریف می کنیم که ورودی های x و y و تعداد epoch ها را دریافت می کند. در ابتدا وزن های لایه پنهان و bias آن مقداری تصادفی دریافت می کنند. سپس وزن های لایه خروجی و bias آن نیز مقداری تصادفی دریافت می کنند.

سپس در هر epoch، برای هر یک از داده های ورودی مراحل زیر انجام میشود:

* خروجی شبکه با تابع predict محاسبه میشود.
* مربع تفاضل مقدار خروجی و مقدار مورد نظر محاسبه شده و به ارور epoch اضافه میشود.
* سپس میزان خطای لایه خروجی حساب می شود. اینکار با تفاضل مقدار خروجی و مقدار مورد نظر و ضرب آن در مشتق خروجی لایه محاسبه میشود.
* سپس میزان خطای لایه پنهان حساب می شود. اینکار با ضرب خطای لایه خروجی در وزن های لایه خروجی در مشتق خروجی لایه پنهان محاسبه میشود.
* میزان تغییر وزن های لایه خروجی با ضرب خطای لایه خروجی در خروجی لایه پنهان محاسبه میشود.
* میزان تغییر وزن های لایه پنهان با ضرب خطای لایه پنهان در ورودی ها محاسبه میشود.
* در آخر وزن ها با توجه به میزان تغییر محاسبه شده تغییر میکنند.

در آخر هر epoch نیز MSE نمایش داده میشود.

## **2-2 تست مدل**

برای تست مدل و استفاده از آن برای XOR ابتدا باید خروجی های تابع XOR را بررسی کنیم.

|  |  |  |
| --- | --- | --- |
| RESULT | B | A |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

اما با توجه به اینکه خروجی یک شبکه نورونی نمی تواند به طور کامل 1 یا 0 باشد، باید برای محسوب کردن یک عدد به عنوان 1 یا 0 محدوده تعریف کنیم. محدوده های تعریف شده به این شکل است:

|  |  |
| --- | --- |
| 0.5 ≤ X < 1 | 0 ≤ X < 0.5 |
| 1 | 0 |

پس از تعریف ورودی ها و خروجی ها در پایتون، یک شی از کلاس DNN با دو نورون پنهان می سازیم. سپس با استفاده از کلاس fit با 2000 epoch شبکه را آموزش می دهیم. نتایج به شکل زیر است:

|  |  |
| --- | --- |
| Predicted | Y |
| 0.12 | 0 |
| 0.90 | 1 |
| 0.87 | 1 |
| 0.11 | 0 |

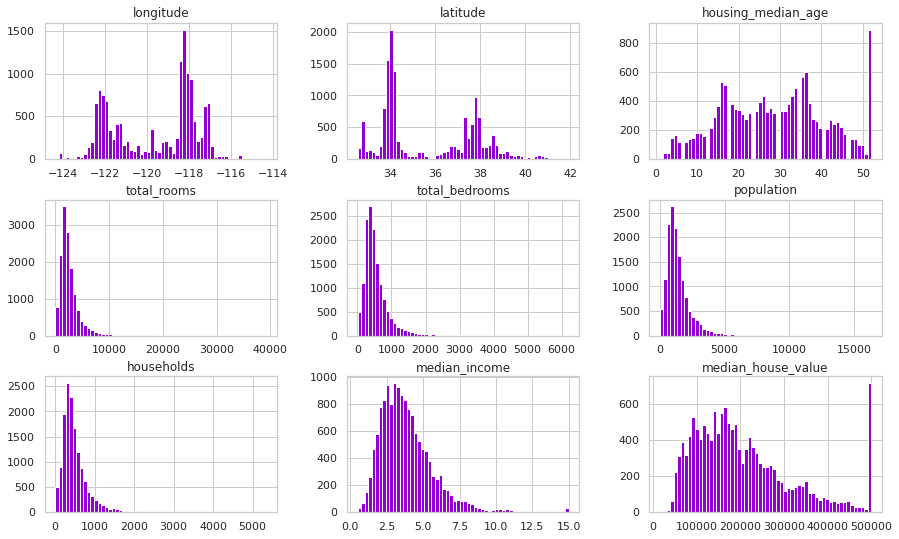
# **فصل سوم، تمرین**

## **3-1 پیاده سازی مدل در محیط متلب**

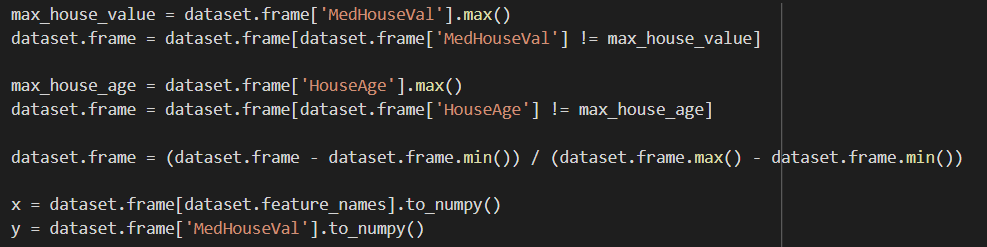
## **3-2 داده های ایستگاه آب و هوایی**

ابتدا با استفاده از تابع fetch\_california\_housing از کتابخانه sklearn دیتاست را load می کنیم. این دیتاست دارای 20640 سطر و 9 ستون است که 8 ستون آن مربوط به feature ها است و یک feature مربوط به target.

ستون های این دیتاست برابر با موارد زیر هستند:

* MedInc: میانگین درآمد در محله
* HouseAge: میانگین سن خانه ها در محله
* AveRoom: میانگین تعداد اتاق ها در محله
* AveBedrms: میانگین تعداد اتاق خواب ها در محله
* Population: میزان جمعیت در محله
* AveOccup: میانگین افرادی که در خانه های محله زندگی می کنند.
* Latitude: طول جغرافیایی محله
* Longitude: عرض جغرافیایی محله

همانطور که از نمودار مقادیر هر ستون مشخص می شود، برخی از داده ها نیاز به حذف شدن دارند، زیرا این داده ها پرت هستند و در نظر گرفتن آنها توسط شبکه می تواند باعث عملکرد ضعیف شبکه شود.

سپس با استفاده از شیوه MinMaxScaling داده ها نرمال میشوند و آماده استفاده توسط شبکه میشود.

همچنین برای جلوگیری از Overfit می توان از شیوه هایی مانند Kfold یا CrossValidation از کتابخانه Sklearn استفاده نمود.

افزایش لایه ها در شبکه باعث افزایش دقت میشود اما زمان بیشتری برای fit نیاز دارد و همچنین با نرخ یادگیری بالا، لایه های بیشتر ممکن است باعث عملکرد ضعیف شوند.

# **منابع و مراجع**

[1] <https://howsam.org/what-is-perceptron-2/>

[2] <https://raahbord.com/neural-network/#%D8%B4%D8%A8%DA%A9%D9%87_%D8%B9%D8%B5%D8%A8%DB%8C_Neural_Network>

[3] <https://blog.faradars.org/%D9%BE%D8%B3-%D8%A7%D9%86%D8%AA%D8%B4%D8%A7%D8%B1/>