به نام خدا

دانشگاه تهران
دانشکده ی مهندسی برق و کامپیوتر

Artificial Intelligance

Computer Assignment 5

علیرضا محمدی	نام و نام خانوادگی
۸۱۰۱۹۵۴۷۱	شماره دانشجویی
۱۳۹۸/۱۰/۱۶	تاریخ ارسال گزارش

۱ پیاده‌سازی شبکه عصبی

در این قسمت می‌خواهیم قسمت‌های مشخص شده در کلاس‌های نورون و لایه‌ی performance را پیاده‌سازی کنیم. این بخش‌ها شامل محاسبه‌ی خروجی هر لایه و همچنین انجام فرآیند backpropagation مربوط به همان لایه است.

در مورد کلاس Input خروجی برابر با مقدار خود این نورون است و گرادیان آن نیز برابر با مقدار صفر قرار داده می‌شود.

در نورون‌های لایه‌های میانی، خروجی هر نورون بر اساس ورودی‌های وزن‌دار و مقدار بایاس هر نورون مشخص می‌شود، به این صورت که ورودی‌های هر نورون به صورت وزن‌دار با هم جمع می‌شوند(ضرب داخلی دو بردار) و در نهایت با مقدار بایاس همان نورون جمع می‌شود. این حاصل جمع در نهایت به یک تابع فعال‌سازی داده می‌شود و بر اساس این تابع خروجی نورون مشخص می‌شود. در این پروژه از تابع سیگموید به عنوان تابع فعال سازی استفاده شده است.

در مورد محاسبه‌ی گرادیان این نود پس از بررسی نود پایانی شبکه عصبی یعنی لایه‌ی performance صحبت می‌شود. لایه‌ی performance به این صورت است که وقتی دیتایی در شبکه‌ی عصبی وارد می‌شود پس از گذر از تمامی لایه‌ها وقتی به نود لایه‌خروجی می‌رسد(با فرض اینکه در این پروژه همیشه یک نود در لایه خروجی داریم) مقدار این نورون لایه‌ی خروجی به این لایه داده می‌شود و در این قسمت با استفاده از تابع پیاده‌سازی شده، performance شبکه برای این دیتا بر اساس کلاس پیش‌بینی شده و مقدار واقعی را محاسبه می‌کنیم. مقدار این تابع برابر با منفی تابع loss است.

در این پروژه از تابع MSE (البته با علامت منفی) به منظور محاسبه‌ی performance شبکه استفاده شده است. معادله‌ی این تابع به صورت زیر است:

$$\text{MSE} = \frac{1}{2}(y - \hat{y})^2$$

خروجی این لایه به این صورت محاسبه می‌شود که مقدار واقعی و پیش‌بینی شده به این تابع داده می‌شود و بر اساس این معادله‌ی بالا خروجی این لایه حساب می‌شود.

حال اگر بخواهیم بحث محاسبه‌ی گرادیان وزن‌های این شبکه را شروع کنیم باید ابتدا از این لایه شروع کنیم زیرا بروزرسانی وزن‌های شبکه با کمک گرادیان تابع performance نسبت به وزن‌های شبکه انجام می‌شود. بر اساس تابع تعریف شده در این لایه و همچنین بر اساس قانون زنجیره‌ای مشتق، گرادیان این تابع بر اساس هر وزن به صورت زیر تعریف می‌شود:

$$\nabla_{w_i} P = (y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i}$$

ترم $(y - \hat{y})$ در واقع گرادیان تابع عملکرد مدل است نسبت به مقدار پیش‌بینی شده و بر اساس قانون زنجیره‌ای، این مقدار در مشتق مقدار پیش‌بینی شده نسبت به وزن مورد نظر، ضرب می‌شود، زیرا در محاسبه‌ی این مقدار پیش‌بینی شده، همه‌ی وزن‌ها اثرگذار هستند.

به منظور محاسبه‌ی ترم دوم گرادیان، در این شبکه باید به صورت بازگشتی عمل کنیم به این صورت که در لایه‌ی آخر، یعنی محاسبه‌ی عملکرد، ترم اول را حساب می‌کنیم و سپس برای محاسبه‌ی ترم دوم، تابع گرادیان لایه‌ی قبلی یعنی لایه‌ی خروجی را صدا می‌زنیم.

خروجی هر لایه به صورت زیر حساب می‌شود:

$$\hat{y} = \sigma(w^T.x)$$

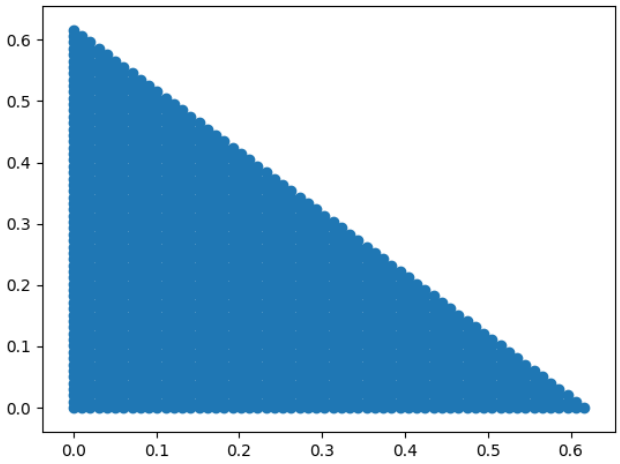
برای محاسبه‌ی گرادیان هر لایه، در مرحله‌ی اول باید گرادیان تابع فعال‌سازی آن لایه را حساب کنیم. نکته‌ی جالبی که در مورد تابع فعال‌سازی sigmoid وجود دارد این است که گرادیان این تابع صورت:

$$f(x)(1 - f(x))$$

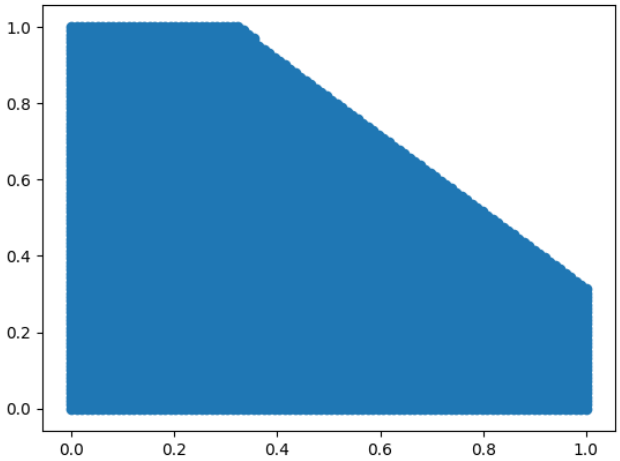
است. و همچنین نکته‌ی مهم این است که ممکن است وزنی که می‌خواهیم گرادیان را نسبت به آن به‌دست آوریم، مربوط به این لایه باشد یا نباشد. در صورتی که مربوط به همین لایه باشد، گرادیان تنها تا همین لایه حساب می‌شود یعنی مقدار x به این وزن وابسته نیست و قانون زنجیره‌ای مشتق در همین لایه متوقف می‌شود، در حالی که اگر این وزن مربوط به لایه‌های پایین‌تر باشد، مقدار x به این وزن وابسته است و در قانون زنجیره‌ای مشتق x نسبت به آن وزن‌هم حساب می‌شود، از جهتی در این حالت، این مقدار که ورودی لایه باشد، در واقع خروجی لایه‌ی قبلی یا $\mathrm{descendant}$ است، پس به صورت بازگشتی، گرادیان لایه‌ی قبلی را صدا می‌زنیم و سپس مقدار آن را در گرادیان این لایه ضرب می‌کنیم، در نهایت مقدار حساب شده به صورت بازگشتی به تابع محاسبه‌ی گرادیان لایه‌ی $\mathrm{performance}$ بر می‌گردد و مقدار گرادیان این وزن محاسبه شده است.

۲ تست کردن شبکه

پس از پیاده‌سازی بخش قبل برای تست عملکرد مدل شبکه‌ی عصبی، آن را روی دیتاست عملگرهای and و or تست می‌کنیم. دقت به دست آمده از آموزش روی هر دو دیتا ۱۰۰ درصد است. همچنین برای مشاهده‌ی ناحیه‌ی تصمیم‌گیری این شبکه‌ی عصبی از تابع پیاده‌سازی در بخش ۷ استفاده شده است و نتایج آن به صورت زیر است:



شکل ۱: Train on OR dataset



شکل ۲: Train on AND dataset

۳ Finite Difference

حال پس از پیاده‌سازی مدل و همچنین تست آن می‌خواهیم از یکی از روش‌های متداول برای عیب‌یابی شبکه‌ی عصبی استفاده کنیم، این روش به منظور تصدیق عملکرد فرآیند backpropagation در شبکه‌ی عصبی مورد استفاده است.

پس از اجرای فرآیند آموزش مدل، مدل را به تابع پیاده‌سازی شده تحویل می‌دهیم تا این روش تایید گرادیان را روی آن اجرا کنیم. ابتدا با استفاده از تکه کد زیر، گرادیان به دست آمده از حل تحلیلی گرادیان را از مدل استخراج می‌کنیم و سپس آن را در آرایه‌ای ذخیره می‌کنیم.

```
for w in network.weights:
   backprop_gradients.append(network.performance.dOutdX(w))
```

سپس می‌خواهیم با استفاده از فرمول ذکر شده در صورت پروژه، گرادیان را تخمین بزنیم و سپس مقدار آن را با گرادیان اصلی مقایسه کنیم، به منظور تخمین هر یک از وزن‌های موجود در شبکه، ابتدا مقدار خروجی تابع performance را بدون هیچ تغییری در مقدار وزن‌ها استخراج می‌کنیم و سپس هر یک از وزن‌ها را با مقدار کوچکی مثل ϵ جمع می‌کنیم و مجددا مقدار خروجی شبکه را حساب می‌کنیم. حال با استفاده از مقادیر ذکر شده که با استفاده از تکه‌کد زیر استخراج می‌شود، گرادیان را برای هر یک از وزن‌ها تخمین می‌زنیم.

```
network.clear_cache()
J = network.performance.output()
J_epsilon = []
for w in network.weights:
   network.clear_cache()
   w.my_value += epsilon
   J_epsilon.append(network.performance.output())
   w.my_value -= epsilon
J_epsilon = np.array(J_epsilon)
gradapprox = (J_epsilon - J) / (epsilon)
print('Approximation', backprop_gradients, gradapprox)
```

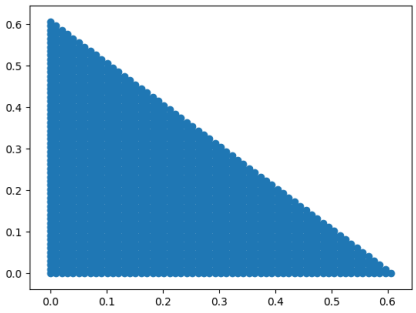
در صورتی که مقدار تخمین زده شده و مقدار واقعی از حد مشخصی به یکدیگر نزدیک‌تر باشند، گرادیان حساب شده در شبکه صحیح بوده است. تصویر پایین نمونه‌ای از اجرای این تابع بر روی شبکه‌ای که روی مدل simple آموزش داده شده است، را نشان می‌دهد.

```
Verifying gradients
Approximation [0.023244386790510795, 0.017433290092883094, -0.023244386790510795] [ 0.02324421  0.01743319 -0.02324456]
The gradient is correct
```

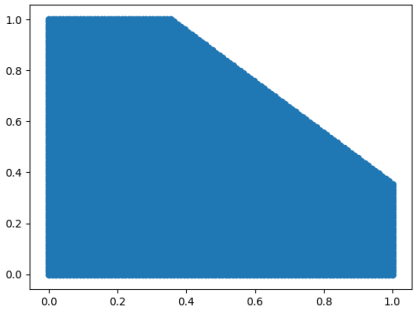
٤ پیاده‌سازی شبکه عصبی دولایه

حال در این بخش می‌خواهیم یک شبکه‌ی دولایه را بر اساس آنچه در دستورکار ذکر شده است طراحی کنیم.
نتایج روی هر دیتاست به صورت زیر است:

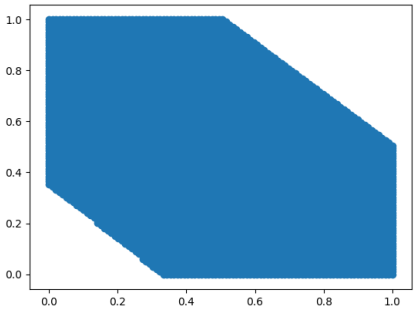
OR دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



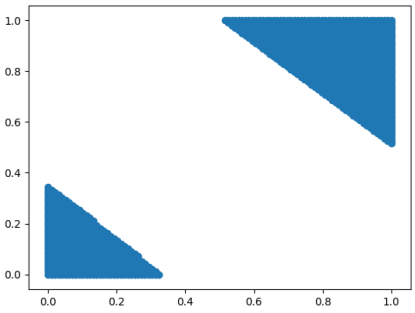
AND دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



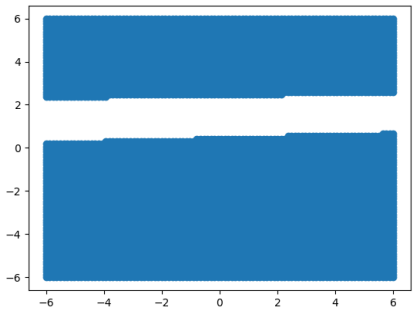
EQUAL دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



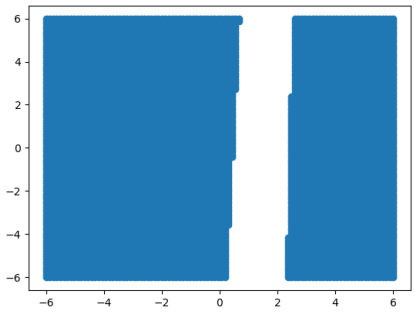
NOT EQUAL دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



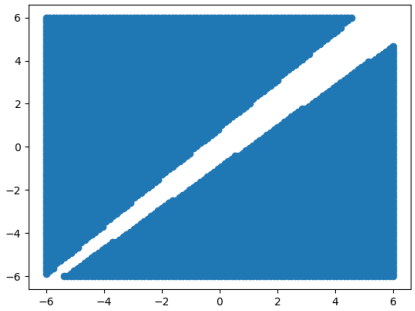
HORIZONTAL دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



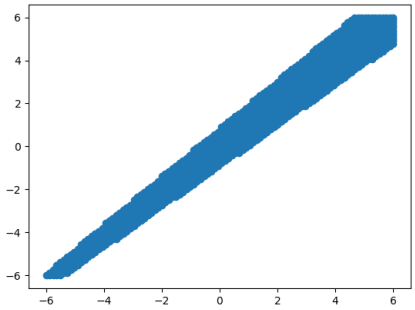
VERTICAL دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



DIAGONAL دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.



INV DIAGONAL دقت حاصل در این دیتاست ۱۰۰ درصد است و همچنین گرادیان‌های حاصل نیز صحیح بوده اند.

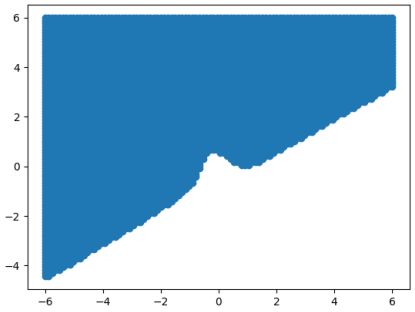


۵ بیش‌برازش

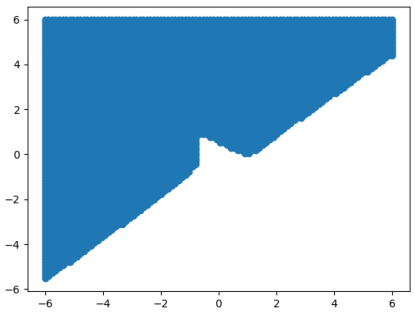
ابتدا مدل توضیح داده‌شده در دستورکار را پیاده‌سازی می‌کنیم و سپس با استفاده از توابع پیاده‌سازی شده این مدل را بر روی دیتاست two-moons آموزش می‌دهیم. دقت حاصل از این مدل به ازای تعداد epoch های مختلف به صورت زیر است:

Test Accuracy	Train Accuracy	Epochs
0.98	0.926829	100
0.94	0.951220	500
0.95	0.951220	1000

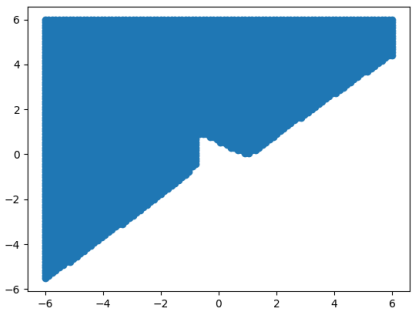
همچنین نواحی تصمیم نیز به صورت زیر است:



شکل ۳: به ازای ۱۰۰ epoch



شکل ۴: به ازای ۵۰۰ epoch



شکل ۵: به ازای ۱۰۰۰ epoch

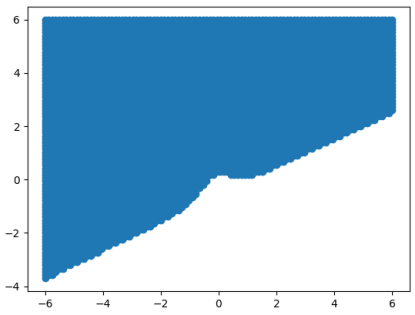
همانطور که مشاهده می‌شود مدل به دلیل پیچیدگی زیاد لایه‌ها (تعداد زیاد نورون‌ها) و همچنین در epoch های بالا، دچار فرابرازش می‌شود و این موضوع به وضوح در ناحیه‌ی تصمیم به دست آمده نیز قابل مشاهده است، به این صورت که ناحیه‌ی تصمیم به خاطر یک سری دیتاها دچار شکستگی شده است و همین شکستگی است که باعث می‌شود هرچند روی دیتاست آموزشی دقت بالاتر می‌رود ولی دقت روی دیتاست تست پایین می‌آید.

حال پس از پیاده‌سازی regularization می‌خواهیم نتیجه‌ی این روش را بر روی این مدل بررسی کنیم و ببینیم که آیا باعث کاهش فرابرازش شده است یا نه؟

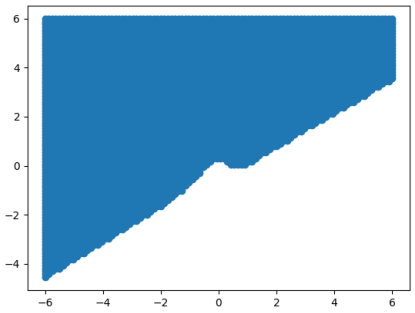
دقت به دست آمده پس از این اصلاح به صورت زیر است: (با انتخاب $\lambda = 0.0005$)

Test Accuracy	Train Accuracy	Epochs
0.97	0.853659	100
0.97	0.853659	500
0.97	0.853659	1000

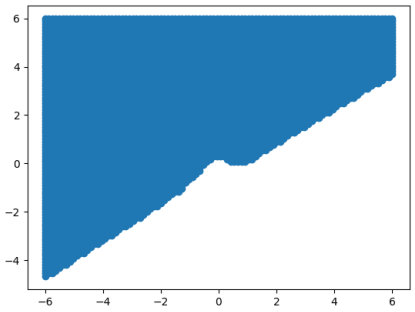
نواحی تصمیم نیز مطابق زیر است:



شکل ۶: به ازای ۱۰۰ epoch



شکل ۷: به ازای ۵۰۰ epoch

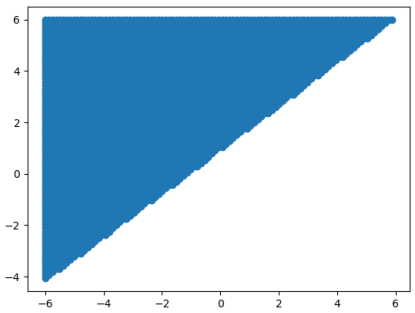


شکل ۸: به ازای ۱۰۰۰ epoch

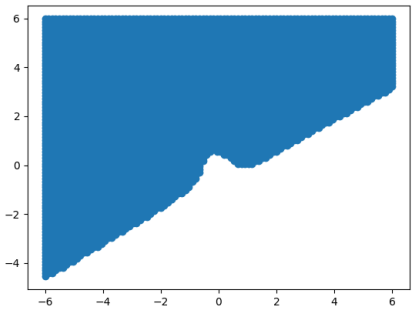
همانگونه که مشاهده می‌شود این ترم regularization که به تابع performance اضافه شده است باعث

شده که فرابرازش مدل روی ناحیه‌ی مرکزی نشان داده شده در شکل بالا کاهش پیدا کند و به جای یک ناحیه تیز [۱] یک ناحیه‌ی نرم به دست آمده است. مطابق جدول بالا دقت مدل به ازای دیتاست آموزشی کاهش پیدا کرده است در حالی که ایجاد این ناحیه‌ی نرم [۲] باعث شده است که مدل روی دیتاست تست، قدرت تعمیم بالاتری داشته باشد.

حال می‌خواهیم ببینیم با افزایش و کاهش مقدار λ چه تغییری رخ می‌دهد.



شکل ۹: به ازای ۱۰۰ epoch و همچنین انتخاب $\lambda = 0.005$



شکل ۱۰: به ازای ۱۰۰ epoch و همچنین انتخاب $\lambda = 0.00005$

همانطور که مشاهده می‌شود با انتخاب λ بزرگ، مدل دچار underfitting می‌شود و باعث می‌شود که مدل توانایی خود را برای طبقه‌بندی دیتاها از دست بدهد در واقع در این حالت بایاس مدل به شدت افزایش یافته است. ولی با انتخاب مقدار خیلی کوچک برای این متغیر، مدل دچار فرابرازش می‌شود و ناحیه‌ی تصمیم‌گیری حتی در epoch های کم نیز تیز می‌شوند و واریانس مدل به شدت بالا می‌رود.

حال می‌خواهیم ببینیم چرا افزودن نرم دوم وزن‌های مدل می‌تواند از فرابرازش آن جلوگیری کند.

در حالت عادی معادله‌ی آپدیت وزن‌ها به صورت زیر است:

$$\omega_i = \omega_{i-1} - \alpha \nabla f$$

[۱] Sharp
[۲] Soft

با افزودن ترم regularization معادله‌ی فوق به صورت زیر در می‌آید:

$$\omega_i = \omega_{i-1} - \alpha \nabla f - 2\lambda \omega_{i-1}$$

برداشت اول این است که فرض کنید که در این آپدیت، گرادیان دارد ما را به سمت فرابرازش می‌برد و کم کردن درصدی از مقدار وزن، باعث می‌شود که از سرعت حرکت به سمت فرابرازش کاسته شود.

برداشت دوم این است که ما به دنبال یک مدل عالی نیستیم، زیرا این مدل عالی تنها روی دیتاست آموزشی عالی عمل می‌کند و قابلیت تعمیم روی دیتاست تست را ندارد. در واقع با افزودن این ترم قصد داریم تا مقداری پنالتی را در آپدیت وزن‌ها در نظر بگیریم که باعث می‌شود مدل به یک مدل عالی تبدیل نشود ولی می‌تواند قابلیت تعمیم بالایی روی دیتاست تست داشته باشد. همانگونه که در جدول بالا مشاهده شد، با افزودن این ترم، دقت روی دیتاست آموزشی کاهش پیدا کرد، یعنی با این ترم پنالتی از یک مدل عالی فاصله گرفتیم ولی باعث شد که دقت روی دیتاست تست بالاتر برود.

از طرفی ترم گرادیان وابسته به دیتا و پارامترهای مدل است و آپدیت وزن‌ها تنها با استفاده از این ترم باعث می‌شود که مدل روی این پارامترها دچار فرابرازش شود در حالی که افزودن یک ترم که شامل پارامتر مستقل λ است باعث می‌شود که از فرابرازش مدل روی دیتا و پارامترها جلوگیری شود.