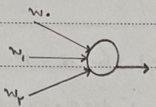
گزارش سوالات:

۱) این سوال طراحی یک پرسپترون ساده است که دارای دو ورودی x_1 و x_2 و یک bias ثابت است و عملیات منطقی NOR را باید learn کند.

x_1	x_2	y
-1	-1	1
-1	1	-1
1	-1	-1
1	1	-1

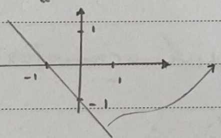


وزن اولیه برای هر کدام از ورودی ها را صفر در نظر میگیریم.

بعد از آن شرط اجرای این الگوریتم آموزش را برابر = کمتر شدن تغییرات از یک حد معین (عددی کوچک) می‌گذاریم و در هر iteration در while برای sample ها آموزش را انجام می‌دهیم و در همین مراحل learning Rate را نیز کاهش می‌دهیم. و برای چک کردن شرط while هم، بیشترین تغییرات را حساب می‌کنیم. در نهایت هم وزن های درست محاسبه شده به ترتیب خروجی نشان داده خواهد شد. (ترتیب جواب را می‌توان با شرط while انشا کرد).

← Stochastic را برای این سوال در نظر گرفتیم و یعنی برای هر sample، weights را update می‌کنیم.

در نهایت بهترین جواب باید به این صورت باشد:



$x + y + 1 = 0$ با پاسخ به دست آمده: $[0.5, -0.5, -0.5]$ ✓

۲) سوال ۲ تا حد بسیار زیادی مشابه سوال اول است و به همان ترتیب باید آموزش را انجام دهیم، اما تعداد sample ها در dataset داده شده است و مانند نمونه قبل ۴ تا نیست اما تعداد ورودی ها مشابه است و تنها ۲ ورودی داریم، کار اضافه ای که در این سوال باید در نظر بگیریم و انجام دهیم، خود تابع Activation function است چرا که در نهایت باید ورودی ها را تقسیم به ۲

Subject:
Date

دسته بندی کنیم. البته نمودار برای میزان خطا در هر مرحله را نیز رسم کنیم.

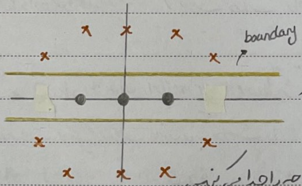
← Activation Function را برای بیرون آوردن تعداد خطا در هر iteration می‌نویسیم و این شرط را هم در نظر می‌گیریم:

$\vec{w}.\vec{x} < 0$ for $d=-1$, $\vec{w}.\vec{x} > 0$ for $d=1$ ✓

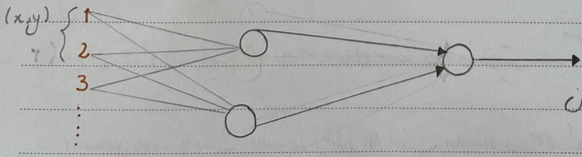
نمودار دقت و خطا نیز رسم می‌شود و می‌توان میزان دقت boundary را مشاهده کرد.

← توضیحات تکمیلی این سوال بعد از پایان سوال ۳.

۳) در روش حل به کمک Madaline در واقع چند Perceptron را با هم خواهیم داشت، Perceptron ها به صورت Parallel و در نهایت خروجی این perceptron ها به یکدیگر وارد می‌شوند (مثلاً And function) نتیجه نهایی را خواهیم داشت. برای مثال ناحیه‌ای که می‌خواهیم تفکیک کنیم مربع است، هر ضلع این مربع یک perceptron است و بعد هم and بین تمام جواب‌های برای هر ضلع.



شکل (آ) برای جواب این شکل باید گفت که حل Madaline می‌تواند این دسته بندی را انجام دهد، به این صورت که اطراف نقاط • مستطیل می‌کشد و این ناحیه را جدا می‌کند.



لایه میانی ۲ boundary مشخص در شکل را برای ما مشخص می‌کنند.
البته این حداقل تعداد نرون است و می‌توان با تعداد بیشتری حل کرد.

← تعداد ورودی‌ها ۲ است فقط (x,y) نقاط داده شده.

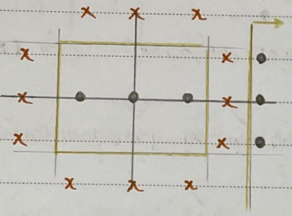
در نهایت بین دو خط، And گرفته می‌شود (توسط نرون لایه آخر) و جواب ناحیه مورد نظر است.

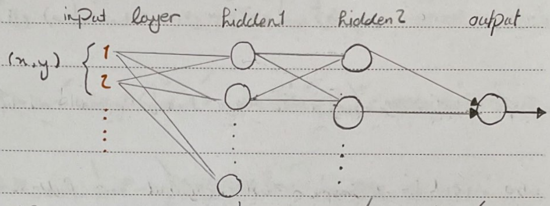
Subject:
Date

شکل ب، همانطور که نشته شده نیز MLP می‌تواند تمام problem ها را solve کند در Theory.

در شکل مثل حدود ۱ لایه مخفی کافی بودن برای مشخص کردن هر خط و خروجی And آنها بود در این سوال نه تنها کشیدن

خطوط، And آنها برای یک ناحیه Convex مورد نیاز است بلکه یک لایه دیگر برای And کردن نواحی Convex لازم است





همانطور که در شکل مشخص بوده ام حداقل 5 نورون در hidden1 برای مشخص خطوط مورد نیاز است. بعد از اینکه خطوط

مشخص شد در لایه بعدی (hidden2) با And این خطوط نواحی Convex را جدا می‌کند و در نهایت در لایه output نواحی

Convex را با هم ادغام می‌کند (And). و تعداد این نورون‌ها نیز برای افزایش سرعت و ... می‌توانند زیادتر شوند ما min در نظر

گرفتیم. ورودی‌های ما تنها (x, y) نقاط هستند و نقاط هم داده‌های ما هستند، تعداد Sample. تعداد neuron های

hidden2 هم در نظرم ۲ تا حداقل است. ← نکته بسیار مهم صفحه بعد **

۲) ادامه سوال ۲: با کاهش varbiable ای که برای شرط while در نظر گرفتن استفاده می‌کنیم (اختلاف تغییرات) دقت

بیشتری را می‌توان کسب کرد. همچنین کاهش آهسته‌تر etha نیز کمک می‌کند که تعداد iteration ↑. تمام این Factor

روی جواب ما تاثیر مستقیم دارد.

PAPCO

Subject:
Date

۳) تمام نکات راه‌حل گفته شده برای این سوال در قسمت "ب" با روش MLP است. اما با روش Madaline این قسمت قابل حل نیست چون که نیاز به یک لایه hidden دیگر هم دارد ولی madaline تنها یک لایه میانی دارد و تنها Convex بودن را مشخص می‌کند و نواحی And Convex را شامل نمی‌شود پس آن سوال با Madaline قابل حل نیست.

۴) روش MLP به طور کامل همراه با مثال (به طور اشتباه) در سوال ۳ قسمت ب توضیح داده شده است. در این روش تعدادی لایه میانی داریم: لایه ورودی که داده‌ها را به لایه hidden1 می‌دهد و این لایه تمامی خطوطی که برای جداسازی لازم است بسته به تعداد perceptron ها می‌کشد. لایه hidden2 وظیفه And کردن در کادر ناحیه Convex بالا از خطوط ایجاد شده دارد و در آخرین لایه output، نواحی And Convex را داریم با این اوصاف هر مسئله‌ای قابل حل است. (Theory)

حال این سوال دسته‌بندی تصاویر اعداد [0-9] است. ورودی‌های ما در اینجا pixel های تصاویر هستند که 28x28 است و ما آن را تبدیل به 784 کردیم برای ورودی. آنها را normalize می‌کنیم (جواب بهتر)، تعداد دسته‌هایی که داریم را مشخص می‌کنیم در اینجا 10 دسته‌بندی مختلف داریم. حال نوبت اضافه کردن لایه‌ها است. آخرین لایه ما شامل 10 neuron است برای Classify کردن، لایه ورودی هم همانطور که گفته شد اندازه 784 دارد. حال در این میان می‌خواهیم لایه hidden قرار دهیم. تعداد لایه‌های میانی، و همچنین تعداد neuron هایی که در هر کدام از این لایه‌ها استفاده می‌شود بر روی دقت نتیجه، همچنین سرعت (به علت افزایش یا کاهش Complexity) تاثیر دارد.

PAPCO

Subject:
Date

input layer ↓ HL ↓ output ↓

مراحل کلی در صفحه قبل توضیح داده شد. همچنین در code هم توضیحات را داریم.

784 { } 10 Classification.

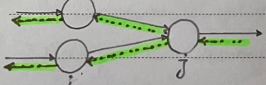
it Could be different number of neurons it depends on our decision.

کتابخانه Keras تمام مراحلی که ذکر شده را خودبه‌خود به‌طور خودکار انجام می‌دهد. در انتها نمودار دقت و خطا را رسم می‌کنیم.

* نکته بسیار مهم همانطور که گفته شد تعیین تعداد لایه‌های پنهانی و تعداد neuron های آن است.

۵) توضیحات مدل کار mlp در سوال قبل مطرح شد حال نوبت اجرای مرحله به مرحله این ساختار است، ابتدا به سراغ محاسبات Back propagation می‌رسیم. همانطور که در توضیحات سوال شش دیدیم می‌توان چند لایه داشت و خطا را محاسبه کرد اما مشکل اینجاست که error لایه‌های پنهانی قابل محاسبه نیست و باید از طریق لایه output که error برای آن قابل محاسبه است back propagate شود.

ابتدا دیدیم forward pass را داریم، پس از آنکه به output layer رسید، error را برای hidden ها می‌سنجیم



به روش زیر:

$$w_{ji} = w_{ji} + \Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

$$v_j = \sum w_{ji} y_i \qquad \frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}} = -\delta_j y_i$$ → local gradient neuron j

$\Delta w_{ji} = \eta \delta_j y_i$ → Compute δ_j for hidden and output layer: (output layer)

$$\delta_j \to -\frac{\partial E}{\partial v_j} = -\frac{\partial E}{\partial e_j} \frac{\partial e_j}{\partial y_j} \frac{\partial y_j}{\partial v_j} = -e_j(-1)\varphi'(v_j) \to e_j \varphi'(v_j) = (d_j - y_j)\varphi'(v_j)$$

برای لایه output به d_j داریم. $e_j(n) = d_j(n) - y_j(n)$

نقش activation function

PAPCO

→ hidden layer: $-\delta_j = -\frac{\partial E}{\partial v_j} = -\frac{\partial E}{\partial y_j}\,\frac{\partial y_j}{\partial v_j} \Rightarrow -\frac{\partial E}{\partial y_j}\cdot \varphi'(v_j)$

بنابراین محاسبه برای لایه‌های میانی به ...

$-\frac{\partial E}{\partial y_j} \Rightarrow$ we will finally have $\sum_{k \text{ in next layer}} \delta_k w_{kj}$

مجموع local gradient لایه‌بعدی مرتبط است.

→ $\Delta w_{ji} = \eta\, y_i\, \delta_j \quad , \quad \delta_j = \begin{cases} \varphi'(v_j)\,(d_j - y_j) & j \text{ output} \\ \varphi'(v_j) \sum_{k \text{ of next layer}} \delta_k w_{kj} & \end{cases}$

همانطور که توضیح داده شد این روش ۲ مرحله forward و backward دارد و در پیاده‌سازی کد هم به همین شکل رفتار شده

و پس از آن وزن‌ها با توجه به نتایج بدست آمده از مراحل قبل update می‌کنیم.

اولین مرحله initialize کردن ماتریس وزن‌ها و ماتریس‌های bias است. ماتریس به این دلیل که چند لایه داریم. و پس از این مراحل

مراحل بالا ✓. در مراحل training در نهایت دقت و خطا را نیز حساب می‌کنیم.

* تمام محاسبات نیز به صورت vectorized بوده است و علاوه بر توضیحات داده شده در اینجا در کد هم به صورت Comment

توضیحات داریم. نیز sigmoid برای تکنیک binary و output layer از softmax برای مسائلی از این

قبیل که چند دسته‌ای است استفاده می‌شود.

← این سوال با لینک داده شده notebook های git زده شده است اما همانطور که توضیحات کامل است به طور خوب متوجه شده‌ام.