سوال اول:

هدف از sparse connectivity درMLPها کاهش اتصالات بین نورون‌ها(وزن‌ها) برای کاهش سایز شبکه، افزایش قابلیت تعمیم دادن به علت کاهش پیچیدگی شبکه و افزایش سرعت یادگیری شبکه است. با کاهش سایز شبکه، این قابلیت به‌وجود می‌آید که بتوانیم مدلمان را در سیستم‌هایی با محدودیت منابعی مانند حافظه هم پیاده سازی کنیم. همچنین این قابلیت برای مدلمان به وجود می‌آید که آسان‌تر و بهتر تفسیر و آنالیز شود و این موضوع می‌تواند در مواقعی مانند بهبود عملکرد مدلمان و تحلیل آن به نفع ما تمام شود.

یکی از تکنیک‌های کاهش اتصالات هرس کردن است که روشهای مختلفی دارد که در ادامه توضیح داده می‌شود. هدف از هرس کردن کاهش اتصالات در عین بهبودی کارایی شبکه است.

انواع روش‌های هرس کردن شامل موارد زیر می‌شوند:

Optimal brain damage(OPD) :

این روش با استفاده از بسط تیلور مرتبه‌ی دو سعی دارد که میزان تاثیر حذف کردن یک وزن یا نورون‌ را بر تابع هزینه محاسبه کند و وزن‌هایی که کمترین تاثیر را دارند پیدا کرده و حذف کند. این روش سعی می‌کند وزن‌هایی با بیشترین تاثیر را دارند، تمرکز بیشتری روی آن‌ها بگذارد ولی بدی این روش هزینه‌ی عملیاتی آن به دلیل محاسبات زیاد آن برای بدست آوردن مشتق دوم است.

Sensitivity-based pruning:

این روش سعی می‌کند تا حساسیت تابع هزینه‌ را بر روی تک تک وزن‌ها حساب کند(با حذف کردن وزن‌ها) تا وزن‌هایی که کمترین تاثیر را روی تابع هزینه دارند و تابع هزینه حساسیت کمی روی آن‌ها دارد را حذف کند. این روش هم مزایای گفته شده را دارد ولی ممکن است به اندازه‌ی روش‌های دیگر کاربردی و موثر نباشد.

Magnitude-based pruning:

این روش سعی می‌کند تا وزن‌هایی با کمترین اندازه را حذف کند. در واقع تمرکز این روش روی اندازه وزن‌ها است(اندازه‌ی قدر مطلق آن‌ها).

مشکل این مدل این است که ممکن‌ است تنها به خاطر پارامتر اندازه، وزن‌های مهم را نیز حذف کند ولی به علت سادگی و سرعت آن، معمولا به عنوان روش اول انتخاب می‌شود.

سوال دوم:

علت اینکه در MLPها باید وزن‌ها را به طور دقیق مقدار دهی کنیم این است که هنگام آپدیت وزن‌ها بتوانیم به سرعت به نقطه ی بهینه برسیم و عملیات یادگیری ما هم بهینه باشد. عدم انجام این کار می‌تواند باعث شود که در نقطه‌ی مینیمم محلی گیر کنیم.

مقداردهی اولیه مناسب وزن‌ها باعث می‌شود که دچار مشکلات vanishing و exploding نشویم.

مقداردهی اولیه متقارن در وزن‌ها(مانند مقداردهی صفر به همه‌ی وزن‌ها) باعث می‌شود که وزن‌های یک لایه خروجی‌های یکسان تولید کنند و به دلیل به طور یکسان و متقارن آپدیت شوند و عملیات یادگیری ما با مشکل مواجه شود و باعث redundency می‌شود. روش‌های زیر استراتژی های جایگزین این روش است.

استراتژی‌های مقدار دهی اولیه‌ی وزن‌ها:

مقداردهی تصادفی:

در این روش وزن‌ها به طور تصادفی که مقدار آن‌ها از توزیع normal یا uniform پیروی می‌کنند مقداردهی می‌شوند. این روش روشی مرسوم در شبکه‌های ساده و کم عمق است ولی در شبکه‌های عمیق می‌تواند باعث مشکل exploding و vanishing شود.

روش xavier:

در این روش وزن‌ها به طریقی مقدار دهی می‌شوند که دارای واریانس زیر باشند:

2/n\_in + n\_out

که n\_in بیانگر تعداد نورون‌های ورودی و دیگری بیانگر تعداد نورون‌های خروجی است.

این روش در مواردی که توابع فعالسازی sig یا tanh است مفید است زیرا واریانس activation ها و Gradient ها در شبیه به هم در طول لایه‌ها نگه می‌دارد.

روش He:

در این روش وزن‌ها به صورت توزیع نرمال و با واریانس زیر انتخاب می‌شوند:

2/n\_in

این روش برای مواردی که تابع فعالیت ما relu است مفید است زیرا به ما کمک می‌کند که از مشکل dying relu جلوگیری کنیم.

سوال سوم:

توابع فعالسازی این قابلیت را به مدل ما اضافه میکنند که بتوانند مدل‌های پیچیده را یاد بگیرند.

تابع sigmoid:

Sig(x) = 1/(1+e^(-x))

Sig’(x) = sig(x)(1-sig(x))

Range = (0,1)

این تابع دچار مشکل اشباع شدگی است، یعنی برای مقادیر بسیار بزرگ یا کوچک دچار مشکل vanishing می‌شود و عملیات gradiant در back propagation کند پیش می‌رود و ممکن است حتی در مینیمم محلی گیر کنیم.

این تابع به دلیل بازه‌ی خروجی اش برای طبقه‌بندی باینری مناسب است ولی در مدل‌های عمیق ممکن است مشکلاتی را به وجود آورد.

به علت non-zero centered بودن خروجی آن، در عملیات آپدیت می‌تواند بهینه نباشد.

تابع tanh:

Tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x) )

Tanh’(x) = 1-tanh(x)^2

Range = (-1,1)

این تابع با این که دچار مشکل اشباع شدگی است ولی به علت zero-centered بودن خروجی اش، در مقایسه با تابع sigmoid می‌تواند دچار مشکل vanishing در عملیات gradiant در back propagation نشود و خیلی سریع تر به نقطه‌ی بهینه می‌رسیم. ولی همچنان به علت مشکل اشباع شدن، برای شبکه‌های عمیق خیلی مناسب نیست.

تابع relu:

Relu(x) = max(0,x)

Relu’(x) = 1 for x>0 and 0 for x=<0

Range = (0,inf)

این تابع به دلیل عدم اشباع شدگی برای مقادیر مثبت، سریع به نقطه‌ی همگرایی می‌رسد.

این تابع به دلیل خروجی صفر برای مقادیر منفی دچار مشکل dying relu می‌شود و برخی نورون‌ها ممکن است غیرفعال شوند و یاد نگیرند. البته این مشکل را می‌توان با مواردی مثل leaky relu و paramwtric relu حل کرد.

به علت سادگی و بهینگی، این تابع، تابع فعالیت پیش‌فرض در لایه‌های پنهان است.

سوال چهارم:

الگوریتم sgd الگوریتمی برای بهینه سازی است که برای آپدیت کردن وزن‌ها از آن استفاده می‌شود.

این الگوریتم بر خلاف الگوریتم bgd که از کل داده‌ها برای آپدیت کردن استفاده می‌کند، تنها از یک نمونه که به طور تصادفی انتخاب می‌شود برای آپدیت کردن وزن‌ها استفاده میکند.

از نظر محاسباتی، sgd بسیار سریع تر از bgd است، زیرا فقط از یک نمونه استفاذه می‌کند و می‌تواند برای دیتاست‌های بزرگتر مفید تر باشد.

الگوریتم bgd خیلی نرم و آرام به سمت نقطه‌ی همگرایی یا بهینه حرکت می‌کند ولی ممکن است در مینیمم محلی گیر کند ولی sgd بسیار نویزی و نا منظم حرکت می‌کند که می‌تواند باعث شود از مینمم محلی بیرون بپریم ولی ممکن است که هیچ‌گاه هم به نقطه‌ی بهینه نرسیم.

الگوریتم mini-bgd از یک زیرمجموعه‌ی تصادفی از نمونه‌ها استفاده می‌کند که یک چیز بین دو الگوریتم بالا است.

هرچقدر سایز mini-batch ما بزرگ یا کوچک باشد، رفتار الگوریتم شبیه bgd یا sgd خواهد شد. مقدار معمول سایز batch بین 32 و 256 است ولی باز این موضوع بستگی به پارامتر‌های مختلف دارد.

سوال پنجم:

L1 regularization:

Loss(L1) = loss(orginal) + sigma(abs(w(i))) from 1 to n

در این نوع منظم سازی، مقادیر برخی وزن‌ها دقیقا صفر می‌شود که باعث کاهش پیچیدگی مدل می‌شود و از بیش‌برازش جلوگیری می‌شود.

L2 regularization:

Loss(L2) = loss(orginal) + sigma(w(i)^2) from 1 to n

در این منظم سازی، مقادیر وزن‌ها دقیقا برابر صفر نمی‌شود بلکه بسیار نزدیک به صفر می‌شود. این موضوع باعث می‌شود که تاثیر وزن به طور کلی از بین نرود که می‌تواند دقت مدل را در عین کاهش پیچیدگی و مشکل بیش‌برازش بالا ببرد.

سوال ششم:

Vanishing:

وقتی گرادیان تابع هزینه در طی عملیات back propagation بسیار کوچک می‌شود این مشکل به وجود می آید. این مشکل در شبکه‌های با تعداد زیادی از لایه‌ها به وجود می‌آید که با ضرب گرادیان در مقادیر کوچک، مقدار نهایی کوچک و کوچک تر می‌شود و به صفر نزدیک می شود.

این موضوع باعث کاهش سرعت می‌شود. همچنین ممکن‌ است برخی فیچر‌های مهم در نظر گرفته نشده و دقت پایین بیاید و در نتیجه یادگیری در شبکه‌های عمیق را چالش برانگیز می‌کند.

Exploding:

این مشکل برعکس مورد قبلی است و به علت مقادیر بسیار بزرگ گرادیان در اثر ضرب‌های مکرر در مقادیر بزرگ حاصل می‌شود که می‌تواند منجر به واگرایی ما از نقطه‌ی بهینه شود.بنابراین ممکن است هرگز به نقطه‌ی بهینه نرسیم و یادگیری مدل ما دچار مشکل شود.

انجام کارهای زیر می‌تواند از vanishing و exploding جلوگیری کند:

مقداردهی اولیه وزن‌ها با تکنیک‌های He, Xavier, random

استفاده از تابع فعالیت مناسب مثل relu

همچنین batch normalization هم می‌تواند کمک کننده باشد

عملیات gradiant clipping و تعیین یک مقدار threshold می‌تواند از رد شدن گرادیان‌ها از یک مرز مشخص جلوگیری کند.

استفاده از معماری‌هایی مثل resNet که دارای residual connection هستند می‌تواند از exploding و vanishing جلوگیری کند.

سوال هفتم:

یکی از راه‌ها استفاده از مقداردهی اولیه با استراتژی‌های xavier و He است که در سوال دوم ریاضیات مربوط به آنها ذکر شده است.

روش دوم تستفاده از توابعی مانند relu برای تابع فعالیت است که به علت خاصیت مشتق آن که در سوال سوم ذکر شده(مقدار ثابت 1 برای مقادیر مثبت) از این دو مشکل جلوگیری می‌کند.

روش بعدی batch normalization است که با نرمال‌سازی ورودی‌های لایه‌ها و stable کردن‌ توزیع انها از این دو مشکل جلوگیری می‌کند.

X\_hat = X-miu/sqrt(var-epsilon)

روش بعدی قرار دادن یک حدبالا مثل T برای جلوگیری از exploding است که به این صورت تعریف می‌شود:

g = (g/norm(g)).T

استفاده از روش‌هایی برپایه‌ی residual connection مانند resNet می‌توان. باعث شود یک‌سری اتصالات را skip کنیم و از مشکلات vanishing جلوگیری کنیم.

y= F(x) + x

سوال هشتم:

مقداردهی همه وزن‌ها به صفر باعث مشکل تقارن می‌شود. این موضوع باعث می‌شود تمام نورون‌ها خروجی یکسان را نتیجه بدهند و در هنگام backpropagation نتایج یکسانی را خروجی می‌دهند و باعث ایجاد مشکل می‌شود. این موضوع باعث می‌شود که تمام نورون‌ها و لایه‌ها ویژگی‌های یکسانی را یاد می‌گیرند و نمی‌توانند ویژگی‌های پیچیده را یاد بگیرند.

مقدار اولیه وزن تصادفی این تقارن را می شکند. با مقداردهی اولیه وزن‌ها با مقادیر تصادفی کوچک، هر نورون با مجموعه‌ای از وزن‌ها شروع می‌کند و به آنها این امکان را می‌دهد ویژگی‌های متفاوتی را از داده‌ها بیاموزند. این تنوع در وزن‌های اولیه، شبکه را قادر می‌سازد تا طیف وسیع‌تری از راه‌حل‌ها را در طول آموزش بررسی کند و توانایی آن را برای یادگیری نگاشت‌های پیچیده از ورودی‌ها به خروجی‌ها بهبود بخشد.

انواع استراتژی‌های وزن دهی داده ها نیز پیشتر با جزئیات در سوالات قبلی بررسی شد، مثل روش xavier, He

سوال نهم:

توابع فعالیت باعث می‌شوند شبکه‌ی ما بتواند خواص غیرخطی را نیز تشخیص داده و یاد بگیرد که بدین ترتیب می‌توانیم نمونه‌های پیچیده را یاد بگیریم. بدون این توابع، مدل ما، فارغ از عمق آن، مانند یک رگرسیون خطی برخورد خواهد کرد.

فرمول‌های ریاضی توابع tanh, sigmoid, relu قبلا بررسی شد در سوال 3.

خواص تابع sigmoid:

گرادیان صاف که به بهینه سازی کمک می کند.

خروجی محدود است و آن را برای طبقه بندی باینری مناسب می کند.

مستعد vanishing برای ورودی های مثبت یا منفی بزرگ، که منجر به یادگیری کند می شود.

موارد استفاده: در لایه خروجی برای وظایف طبقه بندی باینری استفاده می شود.

خواص تابع tanh:

خروجی zero-centered که می تواند به همگرایی سریعتر در مقایسه با سیگموئید کمک کند.

همچنین از مشکل vanishing برای ورودی های بزرگ رنج می برد.

موارد استفاده: مناسب برای لایه های مخفی در شبکه های عصبی، به خصوص زمانی که داده ها حول محور صفر هستند.

خواص relu:

از نظر محاسباتی کارآمد است و به کاهش مشکل vanishing کمک می کند.

می تواند منجر به مشکل dying relu شود، جایی که نورون ها می توانند غیرفعال شوند و همیشه خروجی صفر دارند، چون برای ورودی‌های منفی خروجی صفر است.

موارد استفاده: معمولاً در لایه های پنهان شبکه های عمیق، به ویژه در شبکه های عصبی کانولوشن (CNN) استفاده می شود.

تابع Leaky relu:

Leaky relu(x): x if x>0 and ax if x=<0

a is a very small constant value(typically 0.01)

Leaky relu’(x): 1 if x>0 and a if x=<0

Range: - inf to +inf

خواص این تابع:

این نوع از relu مشکل dying relu را حل می‌کند زیرا در مواقعی که ورودی کوچکتر از صفر است، مشتق مقدار دقیقا صفر ندارد. یک مقدار بسیار کوچک و نزدیک صفر است. این موضوع باعث می‌شود در عین دارا بودن ویژگی‌های relu، معایب ان را میز برطرف کرده باشد.

موارد استفاده: در لایه‌های پنهان در شزکه‌های عمیق استفاده می‌شود تا مشکل dying relu را برطرف کند.

سوال دهم:

تابع softmax یک تابع است که بردارد اعداد حقیقی را به توزیع احتمال تبدیل می‌کند.

Z=[z1, z2, z3,..., zk]

Sigmoid(zi) = e^zi/sigma(e^zj) from 1 to k

عدد k در واقع تعداد کلاس‌ها است و خروجی تابع عددی بین 0 تا 1 است و بردار ما حاوی مقادیر احتمالی است که مجموع انها 1 است.

در مسائل کلاس بندی چندگانه از این تابع در لایه‌ی آخر استفاده می‌شود. بیشترین احتمال در بردار خروجی، کلاس هدف ما را مشخص می‌کند.

تفاوتهاین تابع با sigmoid:

این تابع یک بردار از اعداد بین صفر تا یک بر می‌گرداند ولی تابع sigmoid یک عدد که یا صفر است یا یک برمی‌گرداند.

تابع softmax برای کلاس‌بندی چندگانه استفاده می‌شود ولی Sigmoid برای کلاس بندی دوگانه یا باینری.

از لحاظ فرمول بندی هم تابع softmax یک وکتور از logit ها می‌گیرد و یک وکتور از مقادیر بین 0 نا 1 برمیگرداند ولی Sigmoid یک logit به عنوان ورودی گرفته و یک عدد که یا صفر است یا یک بر می‌گرداند.

سوال یازدهم:

در تابع sigmoid و tanh، مشتق آن وقتی ورودی بسیار بزرگ باشد یا بسیار کوچک باشد، بسیار نزدیک به صفر می‌شود. این موضوع را از روی نمودار‌های این دو تابع می‌توان فهمید.

همانطور که می‌دانیم در backpropagation مشتق‌های توابع با استفاده از قانون مشتق زنجیره‌ای پشت سر هم در هم ضرب می‌شوند:

dL/dw = (dL/da)(da/dz)(dz/dw)

و برای لایه‌های عقب تر این ضرب زنجیره‌ای طولانی تر خواهد شد و ضرب موارد کوچک در هم، منجر به اعداد بسیار کوچکتر خواهد شد که باعث ایجاد این مشکل در عملیات backpropagation در شبکه‌های عمیق می‌شود. این موضوع باعث می‌شود که لایه‌های عقبی در عملیات آپدیت وزن دچار مشکل شوند و یا حتی اصلا آپدیت نشوند. هر چقدر شبکه‌ی ما عمیق تر باشد، این مشکل دردسرساز‌تر خواهد شد.

سوال دوازدهم:

علت سرعت پایین یادگیری در توابعی مانند sigmoid و tanh در ناحیه‌ی اشباع این توابع است که مشتق در آن‌ها بسیار کوچک و نزدیک به صفر می‌شود و آپدیت وزن‌ها خیلی کند اتفاق می‌افتد و رسیدن به نقطه‌ی همگرایی خیلی کند اتفاق می‌افتد و ممکن است اصلا اتفاق نیفتد.

روش‌های مقابله با یادگیری کند:

استفاده از تکنیک‌هایی ماند xavier و he برای وزن‌دهی اولیه که باعث می‌شود توابع فعالسازی در بازه‌ی مناسبی قرار می‌دهد و احتمال قرار گیری در ناحیه‌ی اشباع کاهش می‌یابد

به‌کار گیری batch-normalization می‌تواند فرآیند یادگیری پایدار کند و ورودی‌های هر لایه رو طوری تنظیم کند که در ناحیه‌ی اشباع نیفتیم و یک جریان تقریبا ثابت از گرادیان‌ها را داشته باشیم.

استفاده از تکنیک gradiant clipping باعث می‌شود که از آپدیت‌های بسیار بزرگ جلوگیری شود که از باعث می‌شود آپدیت‌ها از حدی بزرگتر یا کوچکتر نشوند.

استفاده از روش‌هایی مانند RMSprop و Adam که نرخ یادگیری را به صورت پویا آپدیت می‌کند باعث می‌شود تا در نواحی اشباع، سریع تر بتوانیم حرکت کنیم.

استفاده از skip connection مانند ResNet باعث می‌شود تا گرادیان‌ها بسیار بهتر در طول شبکه حرکت کنند و لایه‌هایی که امکان انداختن ما در نواحی اشباع را دارند را رد کنیم.

مزایای تابع ‌های فعالیت جایگزین مثل relu:

تابع فعالیت relu در نواحی مثبت ناحیه‌ی اشباع ندارد و مقدار گرادیان در این ناحیه عدد ثابت یک است که از vanishing و exploding جلوگیری می‌شود.

تابع relu دارای sparsity است(به علت صفر بودن در نواحی منفی) که باعث می‌شود تا تنها یک زیرمجموعه از نورون‌ها فعال شوند و یادگیری ما بهینه تر باشد.

این تابع نسبت به tanh و Sigmoid سریع تر همگرا می‌شود و همچنین از لحاظ محاسباتی هم بهینه تر و هم سریع تر است.

سوال سیزدهم:

علائم بیش‌برازش:

کاهش خطا در مجموعه‌ی training ولی افزایش آن در مجموعه‌ی validation.

افزایش دقت در مجموعه‌ی training ولی کاهش آن در مجموعه‌ی validation.

مدل بسیار پیچیده برای مسائل بسیار ساده(مانند لایه‌ها و نورون‌های بسیار زیاد)

عملکرد بسیار خوب روی مجموعه‌ی آموزشی ولی عملکرد بد روی داده‌های دیده نشده.

حساسیت بسیار بالا به تغییرات

روش مقابله با بیش‌برازش با استفاده از validation set:

در این روش ما داده‌هایمان را به سه بخش train, validation, test تقسیم می‌کنیم که با استفاده از مجموعه‌ی validation ما می‌توانیم عملکرد مدلمان را در طی train بررسی کنیم.

در طی هر epoch ما دقت و هزینه را به صورت نمودار برای هر دو مجموعه‌ی train و validation رسم می‌کنیم و از زمانی که دقت train بالا رفت ولی دقت validation پایین آمد عملیات را متوقف می‌کنیم زیرا این مکان دقیقا مکانی است که به سمت بیش‌برازش حرکت می‌کنیم. برای نمودار هزینه هم همین قضیه صادق است.

هنچنین روش k-fold cross validation داریم که می‌تواند خیلی به ما در این زمینه کمک کند.

سوال چهاردهم:

