دانشکده مهندسی کامپیوتر

مباحث ویژه ۱ (یادگیری عمیق)

تمرین ۶

علی صداقی

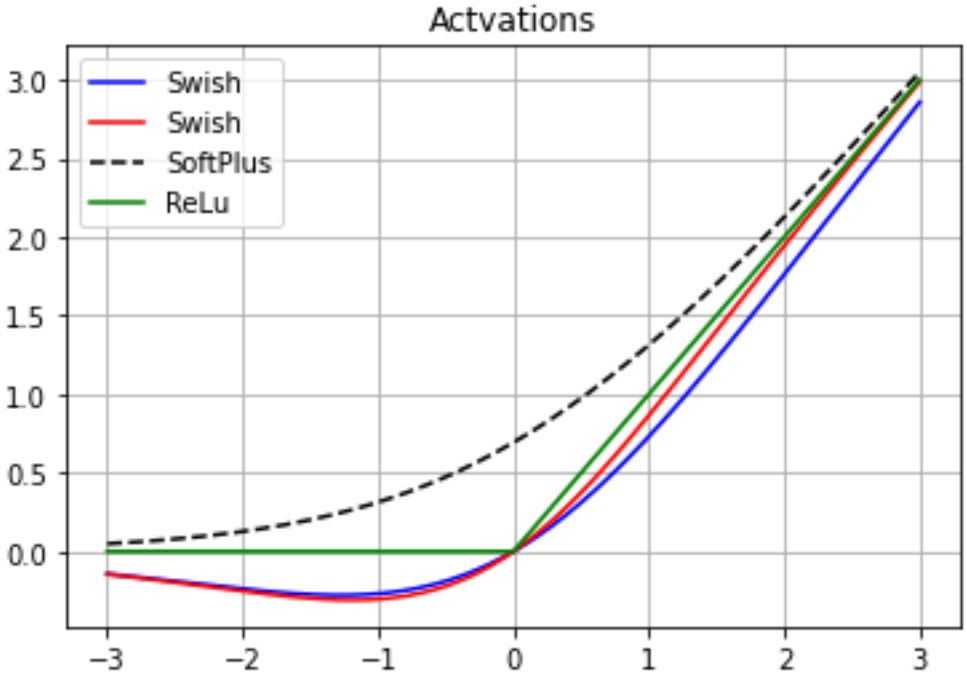
۹۷۵۲۱۳۷۸

سوال ۱) پیاده‌سازی‌ها در نوتبوک موجود است.

الف)

$$Swish: f(x) = x \cdot \sigma(x)$$

$$Mish: f(x) = x \cdot \tanh\big(softplus(x)\big) = x \cdot \tanh(\ln(1 + e^{x}))$$



ب)

$$\frac{dSwish(x)}{dx} = \frac{dx}{dx}\sigma(x) + \frac{d\sigma(x)}{dx}x = \sigma(x) + x \cdot \sigma(x) \cdot \big(1 - \sigma(x)\big)$$

$$= \sigma(x) + x \cdot \sigma(x) - x \cdot \sigma(x)^2$$

$$= x \cdot \sigma(x) + \sigma(x) \cdot \big(1 - x \cdot \sigma(x)\big)$$

$$= Swish(x) + \sigma(x) \cdot (1 - Swish(x))$$

$$\frac{dMish(x)}{dx} = \frac{d}{dx}(x \cdot \tanh(\ln(1 + e^x))$$

$$= \tanh(\ln(1 + e^x)) + x \cdot sech^2(\ln(1 + e^x)) \cdot \frac{e^x}{e^x + 1}$$

$$= \frac{Mish(x)}{x} + x \cdot \frac{e^x}{e^x + 1} \cdot sech^2(\ln(1 + e^x))$$

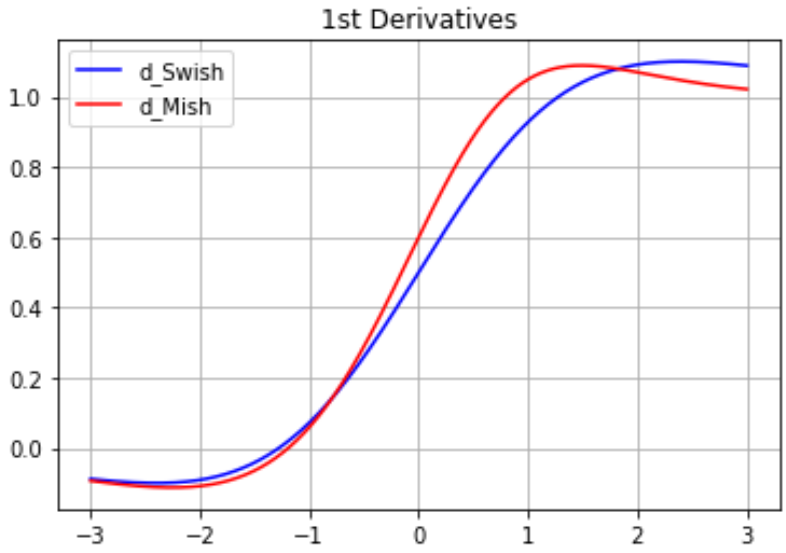
$$= \frac{Mish(x)}{x} + Swish(x) \cdot sech^2\big(softplus(x)\big)$$

با بسط دادن رابطه بالا می‌توان برای مشتق تابع Mish مقدار زیر را محاسبه کرد:

$$\frac{dMish(x)}{dx} = \frac{e^x \omega}{\delta^2}$$

$$\omega = 4(x + 1) + 4e^{2x} + e^{3x} + e^x(4x + 6)$$

$$\delta = 2e^x + e^{2x} + 2$$



ت)

مزیت ReLU نسبت به Sigmoid و Tanh: تابع ReLU به ازای تمامی مقادیر ورودی بزرگتر از صفر مقدار مشتق ثابت ۱ را دارد اما در دو تابع دیگر ذکر شده مشتق به ازای مقادیر بزرگ و کوچک با سرعت افزاینده به صفر میل می‌کند. این تفاوت باعث می‌شود در ReLU کمتر دچار محو شدگی گرادیان (Gradient Vanishing) شویم. همچنین با ReLU سرعت Convergence بیشتر است. (کاهش Saturation به دلیل Unbounded بودن در بالا)

مزیت دیگر ReLU هنگامی است که به ازای Logitهای منفی مشتق صفر می‌شود اما در دو تابع دیگر این طور نیست و مقادیر متقارن غیر صفر تولید می‌شوند. این صفر شدن در ReLU باعث می‌شود تعدادی از نورون‌ها مطلقا خاموش شوند و Sparsity را افزایش می‌دهد در حالی که در دو تابع دیگر نورون‌ها هیچوقت به صورت مطلق خاموش نمی‌شوند و فقط اثراتشان کمرنگ می‌شود (شبکه Dense باقی می‌ماند) (در واقع نوعی از Dropout ایجاد می‌شود)

مزیت دیگر آن محاسبه کمتر در نتیجه سرعت بیشتر است. چون کافیست یک ماکسیمم بین ۰ و مقدار ورودی بگیریم که نسبت به دو تابع دیگر بسیار هزینه کمتری دارد.

- ✓ گونه‌های دیگر ReLU اثرات Inconsistentیی روی دیتاست‌ها و مدل‌های مختلفی دارند و نمی‌توان گفت بهتر هستند.

مشکلات ReLU: با توجه به اینکه مشتق آن برابر ۱ است (در نقاط بزرگتر از صفر) و قاعده مشتق زنجیره‌ای، هیچ راهی برای کوچک کردن مقدار گرادیان وجود ندارد و ممکن است دچار انفجار گرادیان (Gradient Explosion) شویم.

Dying ReLU: اگر ورودی نورون‌های زیادی کوچکتر از صفر باشد مقدار خروجی آن نورون‌ها برابر صفر می‌شود و تعداد زیادی نورون در شبکه خاموش می‌شوند و سایز و قابلیت شبکه برای یادگیری بسیار پایین میاید و نمی‌توان روی داده آموزشی Converge کرد.

مشکل وزن دهی اولیه: اگر وزن‌های اولیه به طور مناسبی مقدار دهی نشوند ممکن است در همان اولین Epoch مقدار Logit نورون‌های زیادی کوچکتر از صفر شود و در نتیجه شبکه در همان ایپاک اول دچار یخ زدگی شود و اکثر نورون‌ها خاموش شوند. همچنین اگر وزن‌ها طوری مقدار دهی شوند که مقادیر Logit بزرگ شود ممکن است شبکه در ایپاک های اولیه دچار انفجار گرادیان شود.

شباهت‌های ReLU، Swish، Mish: هر ۳ این تابع از بالا Unbounded و از پایین Bounded هستند. که اولی باعث جلوگیری از Saturation می‌شود (در مزیت اول ReLU به تفسیر توضیح داده شد). دومی هم باعث خاصیت Regularization می‌شود (در مزیت دوم ReLU به تفسیر توضیح داده شد).

تفاوت‌ها: دو تابع Mish و Swish هر دو Smooth هستند اما ReLU اینطور نیست و در نقطه صفر نیز مشتق پذیر نیست. این امر باعث می‌شود Generalization و Optimization در این دو تابع بهتر باشد.

این دو تابع Non monotonic هستند در صورتی که ReLU تابع صعودی است.

در مقادیر ورودی کوچکتر از صفر ReLU همواره مقدار صفر را خروجی می‌دهد اما در Swish و Mish مقادیر منفی خروجی داده می‌شوند و در کنار ویژگی Non monotonicity باعث می‌شود Expressivity و جریان گرادیان (Gradient Flow) بهتر شود.

ث)

می‌توان تابع Swish را با Reparameterization تبدیل به فرم زیر کرد:

$$f(x;b) = 2x\sigma(\beta x)$$

- ✓ اگر در فرمول بالا از ضریب ۲ صرف نظر کنیم به Swish-B می‌رسیم.

در واقع پارامتر بتا را می‌توان به صورت زیر تفسیر کرد:

اگر مقدار آن برابر صفر باشد: این تابع تبدیل به تابع خطی $f(x) = x$ می‌شود.

اگر مقدار آن به بینهایت میل پیدا کند تابع سیگموید تبدیل به یک تابع *0-1*یی می‌شود و تابع *Swish* مشابه *ReLU* می‌شود.

در واقع اگر بتا را یک پارامتر قابل آموزش در نظر بگیریم، تابع فعال‌سازی نورون با توجه به مقدار بتا می‌تواند در طیفی بین یک تابع خطی *x* و تابع *ReLU* تفسیر شود.

مزیت: با تنظیم شدن پارامتر بتا می‌توان مشخص کرد آن نورون چقدر باید *Regularize* کند مثلا در حالت بتا برابر *0* مقدار ورودی و خروجی یکی هستند و شبکه همان مقدار های *Logit* را خروجی می‌دهد و همواره فعال است (حتی به ازای ورودی منفی) اما در حالت *B* بزرگ شبکه مانند *ReLU* عمل می‌کند و به ازای ورودی‌های منفی نورون را خاموش میکند. هنگامی که بتا برابر ۱ هست همان تابع *Swish* را داریم.

ج)

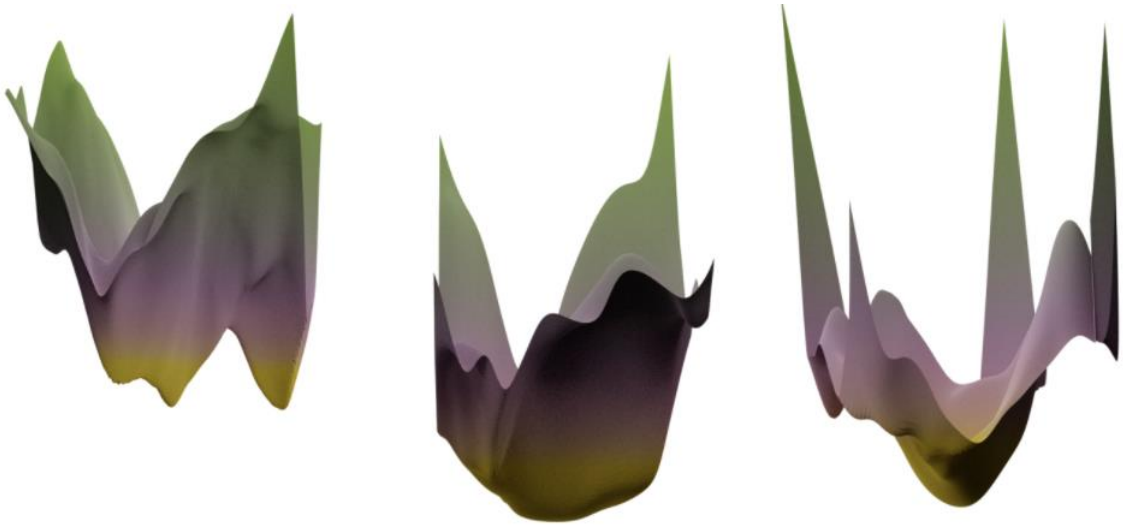
مشتق تابع *Mish* را به صورت زیر به‌دست آوردیم:

$$\frac{dMish(x)}{dx} = \frac{Mish(x)}{x} + Swish(x) \cdot \Delta(x)$$

$$\Delta(x) = sech^2\big(softplus(x)\big)$$

در واقع عبارت دلتا همانند یک پیش‌شرط گذار (Preconditioner) عمل می‌کند و باعث می‌شود گرادیان Smooth تر شود. *Preconditioner* یک بحث مهم در بهینه‌سازی است. در واقع معکوس یک ماتریس متقارن مثبت تعریف شده است و باعث می‌شود هندسه تابع هدف به گونه ای تغییر یابد که *Convergence* روی آن راحت تر رخ دهد. در واقع *Preconditioner* تابعی را که میخواهیم بهینه کنیم را *Smooth* می‌کند تا *Convergence* راحت تر رخ دهد.

همانطور که گفته شد مقدار دلتا در رابطه بالا همان *Preconditioner* است و با فراهم کردن یک اثر قوی *Regularization* باعث می‌شود گرادیان *Smooth* شود در نتیجه بهینه‌سازی راحت تر انجام شود. همین دلتا است که باعث می‌شود *Mish* عملکرد بهتری نسبت به *Swish* در شبکه‌های عمیق و پیچیده داشته باشد.



همانطور که در شکل بالا دیده می‌شود تابع ضرر برای *Mish* (تصویر وسط) بسیار *Smooth* تر می‌باشد نسبت به *ReLU* (تصویر چپ) و *Swish* (تصویر راست). در نتیجه پیدا کردن نقطه مینیمم گلوبال در آن ساده تر است. همچنین احتمال گیر کردن در نقاط زینی و مینیمم محلی کمتر است.

سوال ۲)

الف) با توجه با اینکه با مسئله دسته‌بندی دو کلاسه طرف هستیم برای مقدار Y دو عدد 0 و 1 امکان رخداد است. در هنگام شروع وزن‌های شبکه همگی به صورت Random با توزیع نرمال انتخاب شده‌اند. پس احتمال خروجی به صورت زیر است:

$$P(a=0) = P(a=1) = 0.5$$

فرض میکنیم لیبل واقعی برابر 0 بوده (فرقی با حالت 1 ندارد)

$$BCE(a=0.5, y=0) = -ylog(a) - (1-y)\log(1-a)$$
$$= -\log(1-0.5) = -\log(0.5) = 0.69$$
$$MSE(a=0.5, y=0) = (y-a)^2$$
$$= (0-0.5)^2 = 0.25$$

ب) بیشترین مقداری که هر یک از این توابع خطا می‌توانند تولید کنند در حالتی است که NOT مقدار واقعی پیشبینی شده است. این بیشترین مقدار را برای هر یک از توابع محاسبه می‌کنیم:

$$a = 0, y = 1$$
$$BCE = -\log(0) = +inf$$
$$MSE = (1-0)^2 = 1$$

طبق محاسبات بالا می‌توان گفت مقدار خطا در حالت BCE می‌تواند بسیار بزرگ باشد و چون در حالت Validation احتمال خطا بیشتر از حالت Train است فاصله بین خطا در Validation و Train اختلاف زیادی دارد.

در حالت MSE مقادیر خطای محاسبه شده همواره بین دو عدد 0 و 1 هستند پس حتی در صورت اشتباه هم مقدار خطا خیلی زیاد نمی‌شود در نتیجه دو منحنی Validation و Train نزدیک تر به یکدیگر هستند.

ت) در حالت MSE خطای شبکه بر روی Train و Validation همچنان در حال کاهش است پس باید آموزش را ادامه دهیم یا اگر امکان آن وجود ندارد وزن‌های مربوط به Epoch=100 را برگردانیم.

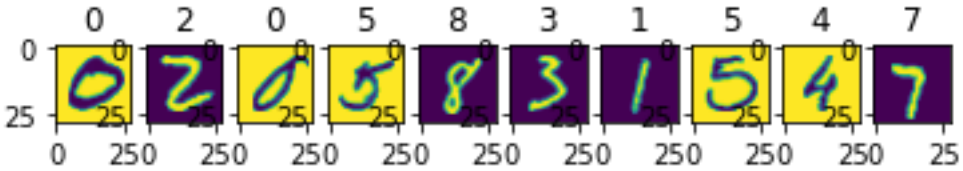
در حالت BCE شبکه دچار Overfit شده است. بهترین عملکرد شبکه زمانی است که خطا روی داده Validation کمینه است. پس باید در Epoch=60 تقریبا باید متوقف شویم.

راه حل درست تر و عملی تر برای توقف استفاده از callbackهای Early Stopping و Model Checkpoint است.

سوال ۳) تمامی خروجی‌ها در نوتبوک موجود است.

بررسی صحت دیتاست: شکل بالا مربوط به Train set، شکل پایین Test set


```
(120000, 28, 28, 1)
(120000, 10)
```




```
(20000, 28, 28, 1)

(20000, 10)
```

بررسی صحت مدل:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 22, 22, 8)         400
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 8)         1608
_________________________________________________________________
flatten_1 (Flatten)          (None, 2592)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                25930
=================================================================
Total params: 27,938
Trainable params: 27,938
Non-trainable params: 0
```

نتایج Leaky ReLU:

alpha	Train	Test
−1.0 = abs	loss: 0.0438 accuracy: 0.9860	loss: 0.0575 accuracy: 0.9828
−0.5	loss: 0.0373 accuracy: 0.9882	loss: 0.0529 accuracy: 0.9833
0.0 = ReLU	loss: 0.0498 accuracy: 0.9845	loss: 0.0621 accuracy: 0.9810
0.5	loss: 0.0799 accuracy: 0.9758	loss: 0.0811 accuracy: 0.9751
1.0	loss: 2.3043 accuracy: 0.1129	loss: 2.3019 accuracy: 0.0868
PReLU	loss: 0.0379 accuracy: 0.9880	loss: 0.0490 accuracy: 0.9843

- Test Loss: PReLU < −0.5 < −1.0 < 0.0 < +0.5 < +1.0
- Test Accuracy: PReLU > −0.5 > −1.0 > 0.0 > +0.5 > +1.0

تفسیر نتایج (بدون در نظر گرفتن PReLU):

در هر دو متریک Loss و Accuracy بر روی داده Test مقدار alpha = −0.5 بهترین نتیجه را داده است. بعد از آن نیز مقدار alpha = −1.0 پس این امر نشان می‌دهد مقادیر منفی برای آلفا بهتر است. دلیل: همانطور که در کلاس درس بررسی شد هنگامی که بخشی از دیتاست نسبت به بخش دیگر آن قرینه شده است تابع فعالسازی قدر مطلق بهترین نتیجه را می‌دهد. زیرا دیگر به ازای ورودی‌های یک دسته (سیاه یا سفید) نورونی خاموش نمی‌شود و شبکه می‌تواند برای تمامی داده‌ها آموزش ببیند در واقع این تابع زمانی مناسب است که علامت z برای ما اهمیت ندارد بلکه بزرگی آن است که برای ما مهم است. دلیل بهتر بودن 0.5− نسبت به 1.0− هم این امر است که در حالت دوم خاصیت Regularization کمتری داریم و نورون‌های بیشتری فعال می‌باشند.

- ✓ مقادیر منفی برای آلفا عملکرد بهتری دارند.
- ✓ هر چه قدرمطلق آلفا کوچکتر باشد عملکرد بهتری داریم.

نتیجه عجیب: در حالت alpha = +1.0 نتایج بسیار بد و عجیب شده. در واقع در این حالت تابع فعالسازی به صورت تابع همانی در می‌آید و همان مقادیر Logit را خروجی می‌دهد. $f(x) = x$

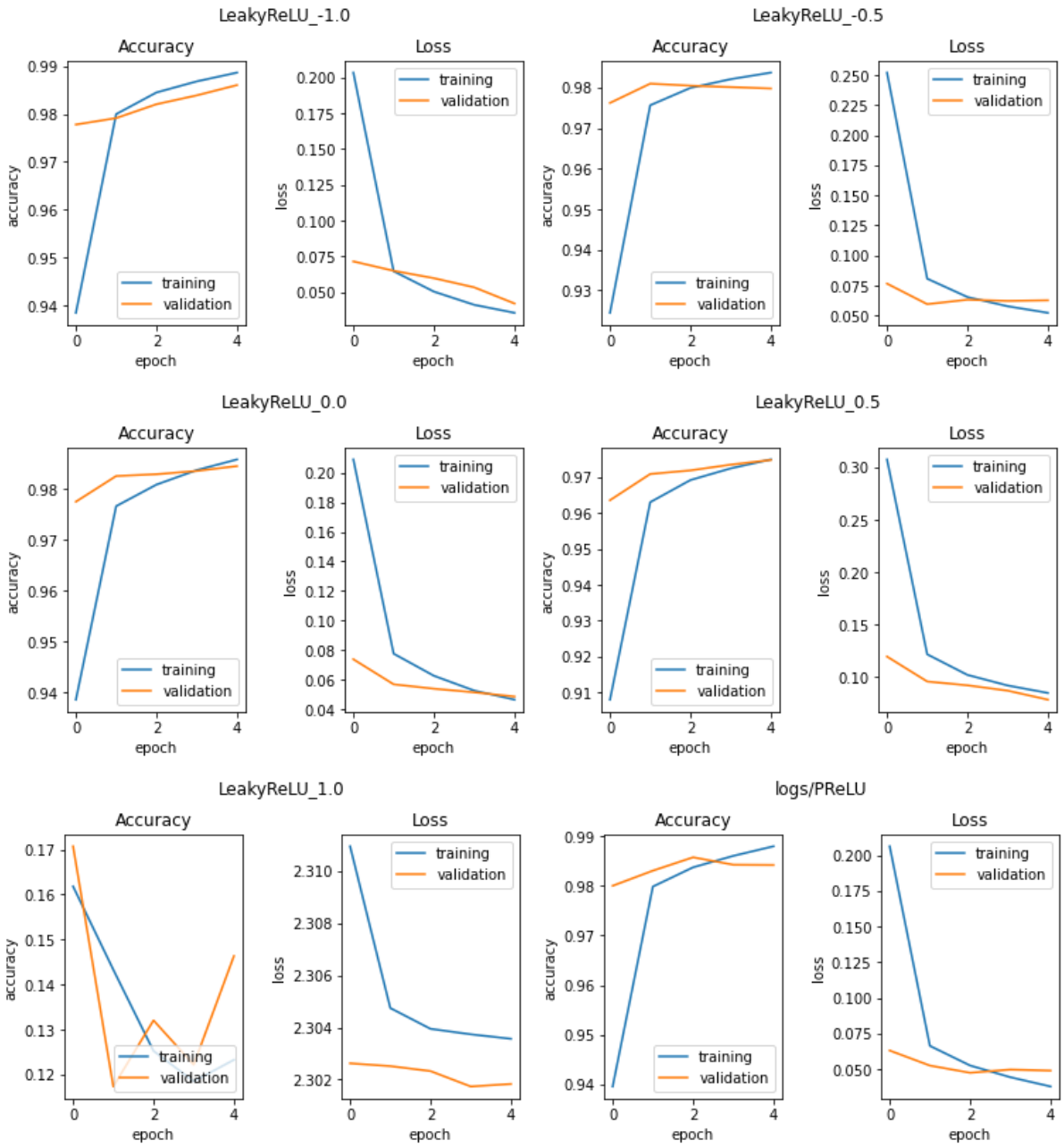
این اتفاق باعث موارد زیر می‌شود:

۱. Linearity در شبکه از بین نمی‌رود در نتیجه شبکه نمی‌تواند ساختارهای پیچیده را یاد بگیرد و همواره محدود به ساختارهای خطی می‌ماند. یعنی در این حالت ما فقط قادر به حل مسائل *Linearly Separable* (با یک خط) خواهیم بود. اما دیتاست *MNIST* تفکیک پذیر خطی نیست.

۲. با توجه به اینکه امکان خاموش شدن (خروجی نورون صفر شود) هیچ نورونی نیست خاصیت *Regularization* نداریم البته این نکته در این مسئله بیان نمی‌شود زیرا اساسا دچار مشکل *High Bias* هستیم نه مشکل *High Variance* و *Overfit*.

۳. اگر تابع *feed forward* شبکه را بنویسیم می‌توانیم آن را با یک شبکه تک لایه و تک نورونه مدل کنیم زیرا حاصل ضرب و جمع چند تابع خطی در نهایت یک تابع خطی می‌شود که همان یک نورون تنهاست.

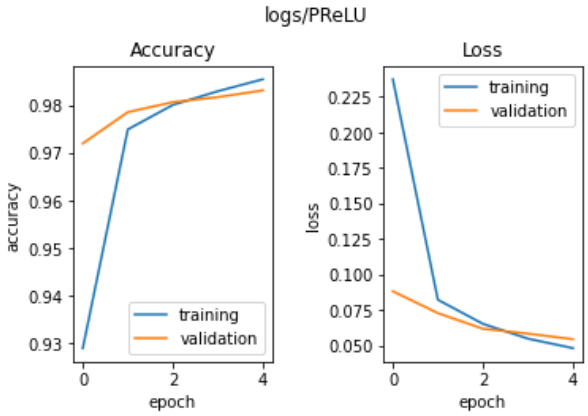
نمودارها:


LeakyReLU_-1.0
Accuracy
Loss
LeakyReLU_-0.5
Accuracy
Loss
LeakyReLU_0.0
Accuracy
Loss
LeakyReLU_0.5
Accuracy
Loss
LeakyReLU_1.0
Accuracy
Loss
logs/PReLU
Accuracy
Loss
training
validation
accuracy
loss
epoch

حالت PReLU:

- Train Loss: 0.0379 Accuracy: 0.9880
- Test Loss: 0.0490 Accuracy: 0.9843

بهترین نتیجه را دارد زیرا تمامی *alpha*ها بهینه می‌شوند.



دسترسی به مقادیر alpha:

```
layer1_alphas = model.get_weights()[2]
layer2_alphas = model.get_weights()[5]

Alphas in 1st Conv2D layer: [[[-0.27639082 -0.33693996 -0.17357634 -
0.13212517 -0.17274912
   -0.21223965  0.3319532  -0.20496829]]]

Alphas in 2nd Conv2D layer: [[[ 0.07129578  0.04566615 -0.00902979 -
0.08019529  0.03539585
   -0.18929955 -0.01344292  0.15101306]]]
```

مورد امتیازی: برای بررسی و مقایسه بهتر ابتدا میانگین و انحراف معیار آلفاها در ۲ لایه را حساب می‌کنیم:

```
Mean of alphas in 1st Conv2D layer: -0.14712952077388763
Mean of alphas in 2nd Conv2D layer: 0.0014254096895456314
STD of alphas in 1st Conv2D layer: 0.19086065888404846
STD of alphas in 2nd Conv2D layer: 0.09600644558668137
```

در لایه اول شبکه مطابق "مقادیر منفی برای آلفا عملکرد بهتری دارند." عمل کرده است اما به جای 0.5- میانگین -0.14 را انتخاب کرده است. زیرا "هر چه قدرمطلق آلفا کوچکتر باشد عملکرد بهتری داریم." با توجه با اینکه ورودی لایه اول خود عکس‌ها می‌باشند و عکس‌ها به دو صورت سیاه یا سفید موجود هستند شبکه تابع فعالسازی را می‌خواسته

که حساس به علامت نباشد و بزرگی Z اهمیت بیشتری داشته است بنابراین مقدار alpha را منفی کرده تا همانند قدرمطلق عمل کند.

در لایه دوم شبکه به سمت میانگین مقدار 0 یا همان ReLU حرکت کرده و سعی کرده مطابق آن عمل کند. زیرا ورودی لایه دوم که خروجی لایه اول می‌باشد دارای ویژگی‌هایی هست که علامت برایش مهم است (چون لایه اول از تصاویر ورودی که حساس به علامت نبودند، ویژگی‌های حساس به علامت تولید کرده و به لایه دوم داده است.)

پس تمامی نکاتی که در تفسیر قسمت Leaky ReLU اشاره کردیم در اینجا نیز اتفاق افتاده است.

✓ از تنسوربورد نیز استفاده شد اما چون کیفیت تصاویر matplotlib بهتر بود از آن برای تصویر برداری نمودار ها استفاده کردیم.