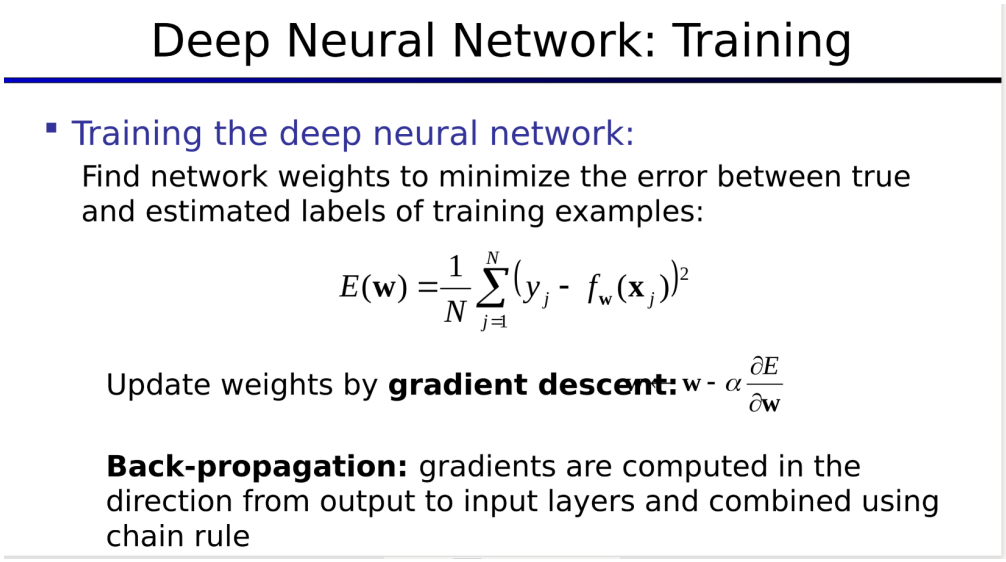
گزارش‌کار پروژه شبکه عصبی درس هوش مصنوعی

توجه: در این‌جا صرفاً پاسخ سوالاتی که در صورت پروژه ذکر شده آورده شده است. توضیحات مربوط به کد در فایلی جداگانه قرار دارد.

پرسش یکم

- اگر مقدار اولیه تمام وزن‌های شبکه برابر صفر قرار بود و شبکه را آموزش می‌دادید، نتیجه آن چه بود؟ خروجی را با حالت قبل (که در آن وزن ها به صورت مقدار تصادفی مقداردهی اولیه می‌شدند)، مقایسه کنید. نیازی به پیاده سازی نیست.

درصورتی که وزن اولیه تمام نورون‌ها در شبکه برابر با صفر باشد، در هر مرحله نورون‌ها همان ویژگی‌هایی را ارزیابی می‌کنند که در مرحله قبل می‌کردند. به این معنی که featureای که بر اساس آن داده‌ها را تقسیم‌بندی می‌کنند ثابت می‌ماند. دلیل این مسئله این است که مقدار derivative به هیچ وجه تغییر نمی‌کند چرا که هر بار مقداری که باعث تغییر آن می‌شود در صفر ضرب می‌شود.



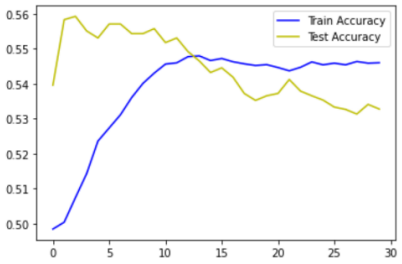
پرسش دوم

یکی از پارامتر های مهم در آموزش دادن شبکه های عصبی، learning rate می باشد.

- حال با کاهش و افزایش این پارامتر، شبکه را آموزش دهید و پس از یافتن مقدار بهینه برای شبکه خود، نتیجه را گزارش کنید.
- همچنین رفتار شبکه را برای learning rate با مقدار بالاتر (10 برابر) و پایین‌تر (0.1 برابر) نسبت به حالت قبل را بدست آورید. نتیجه خود را با حالت قبل مقایسه کنید و توجیه کنید.

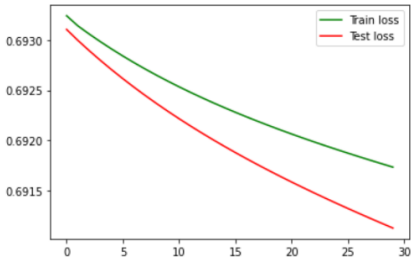
برای تمام قسمتهای بعد، از learning rate بهینه‌ای که به دست آورده‌اید استفاده کنید.

همان‌طور که در نمودارهایی که در کد نشان داده شده است می‌توان دید، درصورتی که نرخ یادگیری خیلی زیاد یا خیلی کم باشد، دقت مدل ما به‌روزرسانی نشده و یا در یک محدوده گیر می‌کند.




```
In [72]:
1 train_loss, = pplt.plot(log['train_loss'], label="Train loss", color='g')
2 test_loss, = pplt.plot(log['test_loss'], label="Test loss", color='r')
3 pplt.legend(handles = [train_loss, test_loss])
```

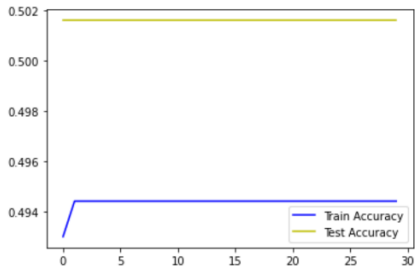
```
Out[72]: <matplotlib.legend.Legend at 0x7fd77efc5670>
```



توجه داشته باشید که learning rate را اگر در مثال کوهنوردی ترجمه کنیم، به معنی اندازه گام‌هایی است که می‌توانیم برداریم. درصورتی که این مقدار خیلی کم باشد، یا خیلی دیر به مقدار ماکسیمم یا مینیمم محلی می‌رسیم (میل کردن به حد تابع دیر اتفاق می‌افتد) یا این که در یک نقطه paddle گیر

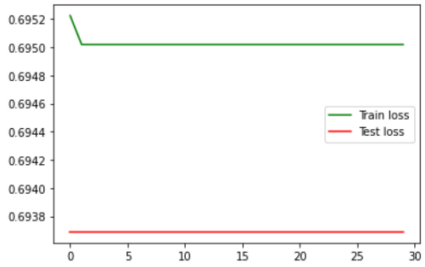
می‌کنیم. در این‌جا همین‌طور که دیده می‌شود انگار در یک نقطه گیر کرده‌ایم و حتی دقت داده تست کمتر هم می‌شود.

```
Out[88]: <matplotlib.legend.Legend at 0x7fd764cf5310>
```




```
In [89]:
1 train_loss, = pplt.plot(log['train_loss'], label="Train loss", color='g')
2 test_loss, = pplt.plot(log['test_loss'], label="Test loss", color='r')
3 pplt.legend(handles = [train_loss, test_loss])
```

```
Out[89]: <matplotlib.legend.Legend at 0x7fd775f8a880>
```



در صورتی که مقدار learning rate خیلی زیاد باشد هم انگار که گام‌های خیلی بزرگی انتخاب کردیم و فرایند یادگیری (به عبارتی میل کردن به حد تابع در یک نقطه) اصلاً اتفاق نمی‌افتد. در این‌جا واضح است که هیچ یادگیری‌ای نداریم. چرا که نمودارها از یک جایی به بعد مقدار قابت پیدا می‌کنند و این عملاً به آن معنی است که هیچ یادگیری‌ای اتفاق نمی‌افتد.

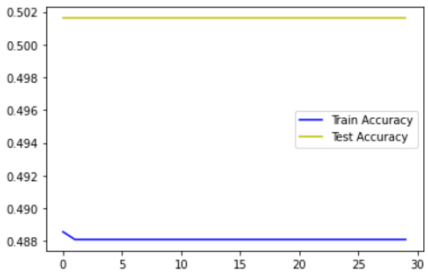
پرسش سوم

- عملکرد شبکه‌ی طراحی شده در قسمت اول را به کمک Activation Function های زیر بسنجید و نتایج را مقایسه نمایید.
  - تابع فعال‌ساز Sigmoid
  - تابع فعال‌ساز Hyperbolic Tangent
  - تابع فعال‌ساز Leaky ReLU
- دلیل اینکه Tanh و Sigmoid عملکرد مناسبی برای این دست شبکه‌ها ندارند را بیان کنید.
- برتری Leaky Relu نسبت به Relu چیست؟

توجه: در ادامه مراحل، از activation function با بهترین نتیجه در لایه‌های شبکه استفاده نمایید.

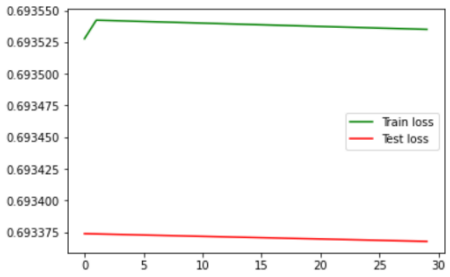
تابع سیگموید

Out[92]: <matplotlib.legend.Legend at 0x7fd7650b1b20>



In [93]:
```
train_loss, = pplt.plot(log['train_loss'], label="Train loss", color='g')
test_loss, = pplt.plot(log['test_loss'], label="Test loss", color='r')
pplt.legend(handles = [train_loss, test_loss])
```

Out[93]: <matplotlib.legend.Legend at 0x7fd75396b970>



استفاده از تابع سیگموید در مقایسه با ReLU شبیه به زمانی عمل می‌کند که نرخ یادگیری را خیلی زیاد در نظر بگیریم. هر چقدر ورودی ما اندازه بزرگ‌تری داشته باشد، گرادیان تابع سیگموید کمتر می‌شود. اما دلیل دیگری هم وجود دارد که چرا نمودار دقت داده‌های تست و آموزش به صورتی که در تصویر دیده شد در می‌آید؛ مقدار derivative تابع سیگموید! مقدار این تابع همواره کمتر از ۰٫۲۵ بوده که باعث می‌شود تاثیری شبیه به زمانی داشته باشد که قدم‌های ما تاثیری در یادگیری شبکه عصبی ندارند. اگر این نکته را هم در نظر بگیریم که در شبکه عصبی چندین لایه با این تابع فعال‌ساز داشته باشیم، این مقدار کوچک derivative متناوباً در خروجی‌های نورون‌ها ضرب شده و مقدار خروجی به صفر میل می‌کند.

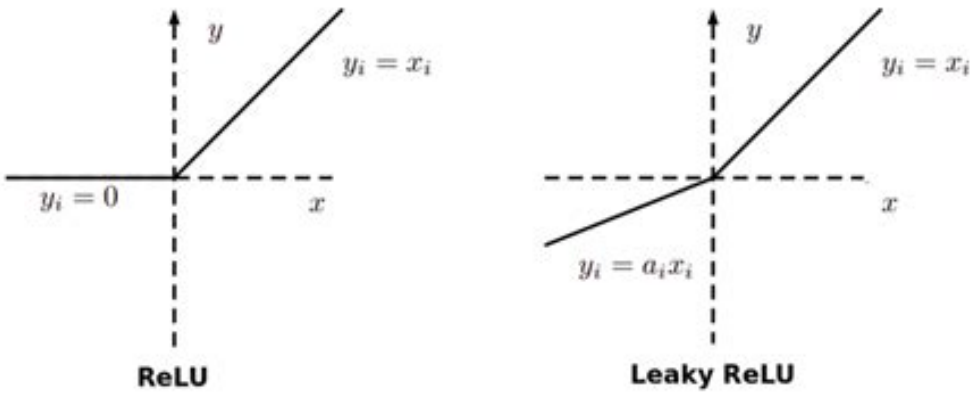
تابع تانژانت هایپربولیک

دلیل این که این تابع به کار نمی‌آید (دست کم در کدی که من نوشتم) این است که سربار محاسباتی زیادی دارد.

```
def update_weights(self, backprop_tensor, lr):
    '''
    It updates Layer weights according to the backpropagation matrix and learning rate.
    This method updates bias values as well.
    Parameters:
        backprop_tensor: 2d np.matrix passed from the next layer containing gradient values.
        lr: learning rate
    Returns:
        backprop_tensor to be used by the previous layer.
    '''
    assert np.ndim(backprop_tensor)==2
    assert np.size(backprop_tensor,0) == np.size(self.__last_activation_derivative,0)
    assert np.size(backprop_tensor,1) == self.__n_neurons

    transpose_input_matrix = self.__last_input.transpose()

    backprop_mult = np.multiply(backprop_tensor, self.__last_activation_derivative)
    backprop_matrix = np.matrix(np.tile(1, (1, backprop_mult.shape[0])))

    weight_product = np.matmul(transpose_input_matrix, backprop_mult)
    backprop_product = np.matmul(backprop_matrix, backprop_mult)

    backprop_tensor = np.matmul(backprop_mult, self.__weight.transpose())

    self.__weight -= weight_product * lr
    self.__bias -= backprop_product * lr

    return backprop_tensor
```

در این قسمت از کد که وزن‌ها را به‌روزرسانی می‌کنیم، توابعی مثل matmul که ضرب ماتریسی را انجام می‌دهند، با overflow مواجه می‌شوند. همچنین درمورد این تابع نیز با مشکل Gradient Vanishing مواجه هستیم. در این لینک توضیحات بیشتر درمورد این پدیده وجود دارد. توابعی مثل سیگموید یا همین تانژانت بازه‌ی نسبتاً قابل‌توجهی از ورودی را به بازه‌ی کوچکی از خروجی map می‌کنند. این کار باعث می‌شود که variance مقادیری که در شبکه عصبی باعث یادگیری می‌شوند کاهش پیدا کرده و همچنین اثر آن‌ها در یادگیری شبکه نیز از بین برود.

تابع Leacky ReLU



در کد ما Leacky ReLU و ReLU عملکردی نسبتاً مشابه دارند. اما به طور کلی Leacky ReLU دو برتری نسبت به حالت عادی خود دارد. یکی این که در آن با مشکل vanishing gradient روبرو نمی‌شویم. یکی دیگر این که با مشکل Dead ReLU که در آن مقادیر کوچک x کلاً مقدار صفر دارند روبرو نمی‌شویم. از طرف دیگر در مقایسه با سیگموید و تانژانت سربار محاسباتی کمتری دارد و به طور واضحی سرعت محاسبات را بیشتر می‌کند.

پرسش چهارم

قسمت پنجم) تاثیر batch size

- عملکرد شبکه را به ازای با batch size مقدار 16 و 256 بدست آورید. نتیجه خود را با حالت قبل مقایسه کنید و توجیه کنید.
- علت استفاده از batch در فرایند آموزش چیست؟ مزایا و معایب احتمالی batch size بسیار کوچک را شرح دهید.

به طور کلی داشتن batch به ما این امکان را می‌دهد که در آن واحد بتوانیم محاسبات بیشتری انجام دهیم و مکانسیم موازی‌سازی را برای ما فراهم می‌کند. استفاده از batch کوچک‌تر باعث می‌شود که همگرایی به سمت حد تابع سریع‌تر اتفاق بیفتد. می‌توان این‌طور استدلال کرد که مدل ما قبل از این که با حجم زیادی از داده در یک زمان روبرو شود، فرصت یادگیری دارد. هم‌چنین اگر batch خیلی بزرگ باشد، به احتمال خیلی زیادی در نقطه بهینه محلی گیر نمی‌کنیم و ما ار به نقطه بهینه عمومی می‌رساند (دست‌کم برای توابع محدب) اما این به قیمت کاهش generalization مدل تمام می‌شود.
معمولاً پیشنهاد می‌شود که شروع یادگیری با اندازه batch کوچک باشد که مدل بتواند یادگیری خود را انجام دهد و خیلی پرت نشود. رفته‌رفته سایز batch بیشتر شود تا بتوانیم با احتمال بیشتری به سمت نقطه بهینه عمومی حرکت کنیم.