بسمه تعالی

دانشکده مهندسی کامپیوتر

یادگیری عمیق

نام استاد: دکتر محمدی

تمرین هفتم

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

آبان ۱۴۰۰

۱. پاسخ سوال اول

Overfit: مدل هایی که دچار overfit شده‌اند دارای bias کم و variance بالا هستند. مشکل در این مدل‌ها میتواند پیچیدگی زیاد مدل نسبت به دیتاست باشد پس یکی از راه حل ها طبیعتا استفاده از مدل ساده تر (استفاده از لایه های کمتر یا نورون های کمتر در هر لایه) باشد. همچنین علت overfit شدن مدل می تواند حفظ شدن دیتا ها باشد. مثلا فقط ۱۰۰ عکس گربه و سگ در داده آموزشی داریم و با شبکه‌ای به بزرگی resnet میخواهیم آن ها را تفکیک کنیم! پس راه حل مناسب دیگری میتواند بزرگتر کردن دیتاست باشد. در این راه میتوان از data augmentation استفاده کرد. راه حل دیگری که در شبکه های عمیق به شدت استفاده میشود استفاده از لایه dropout است که با استفاده از آن هر بار درصدی از لایه های میانی را از تصمیم گیری حذف میکنیم، با این کار لایه ها نسبت به نویز مقاوم تر شده و از overfitting هم جلوگیری میشود. همچنین الگوریتم‌های مختلف regularization هم برای جلوگیری ازین مشکل استفاده میشوند. مثلا استفاده از l1 یا l2 به عنوان kernel regularizer میتواند راهگشا باشد.

Underfit: مدل هایی که دچار underfitting هستند دارای bias زیاد و variance کم هستند. درواقع برعکس حالت قبل می باشد. پس وقتی مدل ما دچار این معضل می شود احتمالا مدل برای تفکیک یا حل چنان دیتاستی خیلی ساده است و بدیهی ترین راه برای حل این مشکل پیچیده تر کردن مدل با زیاد کردن لایه ها یا نورون های هر لایه یا بزرگ کردن فیلتر های لایه کانولوشنی و ... است. همچنین ممکن است تعداد ویژگی هایی که از هر نمونه در دیتاست داریم برای تفکیک آن کافی نباشد و به همین خاطر به دقت خوبی نمیرسیم. پس یک راه حل افزودن feature هاست. یا ممکن است آموزش مدل به اندازه کافی انجام نشده باشد که زیاد کردن epoch های آموزش این مشکل را رفع میکند تا شبکه همگرا شود.

۲. پاسخ سوال دوم

فرض ها ← تابع فعالسازی لایه آخر relu ، learning rate=0.1 ، alpha=0.01
beta1=0.9 : beta2 = 0.999

forward Propagation formulas:

$$h_1 = Relu(i_1 w_1 + i_2 w_2)$$

$$h_2 = Relu(i_1 w_3 + i_2 w_4)$$

$$\hat{Y} = Relu(h_1 w_5 + h_2 w_6)$$

backward propagation:

محاسبات را برای یک داده انجام می دهیم تا روابط را به دست آوریم. اما هنگام آپدیت کردن وزن ها باید گرادیان ها را میانگین تمام داده‌ها (در اینجا 2 تا) لحاظ کنیم.

$$\frac{d\,loss}{d\hat{Y}} = -2(Y-\hat{Y}) = 2(\hat{Y}-Y)$$ ← مقدار label

$$\frac{d\hat{Y}}{d(h_1 w_5 + h_2 w_6)} \xrightarrow{relu\ derivative} \begin{cases} 0 & x<0 \\ 1 & x>0 \end{cases}$$ ← مشتق ورودی relu

$$\frac{d(h_1 w_5 + h_2 w_6)}{dh_2} = w_6 \quad , \quad \frac{d(h_1 w_5 + h_2 w_6)}{dh_1} = w_5$$

$$\frac{dh_1}{dw_1} = i_1 \times \begin{cases} 0 \\ 1 \end{cases} \quad \frac{dh_1}{dw_2} = i_2 \times \begin{cases} 0 \\ 1 \end{cases} \quad \frac{dh_2}{dw_3} = i_1 \times \begin{cases} 0 \\ 1 \end{cases} \quad \frac{dh_2}{dw_4} = i_2 \times \begin{cases} 0 \\ 1 \end{cases}$$

حالا با استفاده از قانون مشتق زنجیره‌ای روابط آپدیت وزن‌ها را به دست می‌آوریم:

$$\frac{dloss}{dw_6} = \frac{dloss}{d\hat{Y}} \times \frac{d\hat{Y}}{d(h_1w_5+h_2w_6)} \times \frac{d(h_1w_5+h_2w_6)}{dw_6} = 2(\hat{Y}-Y) \times (1 \text{ یا } 0) \times h_2 + \alpha w_6$$

regularization

به طور مشابه $\rightarrow \frac{dloss}{dw_5} = 2(\hat{Y}-Y) \times (1 \text{ یا } 0) \times h_1 + \alpha w_5$

$$\frac{dloss}{dw_4} = 2(\hat{Y}-Y) \times \frac{d\hat{Y}}{d(h_1w_5+h_2w_6)} \times w_6 \times \frac{dh_2}{d(i_1w_3+i_2w_4)} \times i_2 + \alpha w_4$$

$d\hat{Y}/d(\cdots)$: 1 یا 0 — $dh_2/d(\cdots)$: 1 یا 0

به طور مشابه $\rightarrow \frac{dloss}{dw_3} = 2(\hat{Y}-Y) \times w_6 \times (1 \text{ یا } 0) \times (1 \text{ یا } 0) \times i_1 + \alpha w_3$

$$\frac{dloss}{dw_2} = 2(\hat{Y}-Y) \times w_5 \times (1 \text{ یا } 0) \times (1 \text{ یا } 0) \times i_2 + \alpha w_2$$

$$\frac{dloss}{dw_1} = 2(\hat{Y}-Y) \times w_5 \times (1 \text{ یا } 0) \times (1 \text{ یا } 0) \times i_1 + \alpha w_1$$

حالا که روابط را به دست آوردیم، باید با روابط Adam آن‌ها را آپدیت کنیم.

نکته‌ای که باید توجه کنیم این است که در روابط مشتق به علت جمع شدن $\frac{1}{2}w_i^2$ با تابع ضرر، برای هر وزن ما αw_i هم موقع مشتق گیری داریم که چون به عمل جمع بسته بنابت در روابط زنجیره‌ای آن را ننوشتم و در رابطه نهایی آوردم.

$$moment_{first} = \beta_1 \times moment_{first} + (1-\beta_1)\, dX$$

$$moment_{second} = \beta_2 \times moment_{second} + (1-\beta_2)\, dX$$

$$unbias_{first} = \frac{moment_{first}}{1-\beta_1^2}$$

$$unbias_{second} = \frac{moment_{second}}{1-\beta_2^2}$$

$$w = w - \frac{unbias_{first}}{\sqrt{unbias_{second}}}$$

اگر وزن ها را با مقادیر $w_5=0.5$ ، $w_4=0.4$ ، $w_3=0.3$ ، $w_2=0.2$ ، $w_1=0.1$ و $w_6=0.6$ مقدار دهی کنیم ، مقادیر loss و وزن ها پس از هر epoch : (جلسات با ماشین حساب)

epoch 0 ⟶ mean squared error = 104.0576

epoch 1 ⟶ " " " = 45.6462

$$\begin{cases} w_1 \simeq 0.2 \\ w_2 \simeq 0.3 \\ w_3 \simeq 0.4 \\ w_4 \simeq 0.5 \\ w_5 \simeq 0.6 \\ w_6 \simeq 0.7 \end{cases}$$

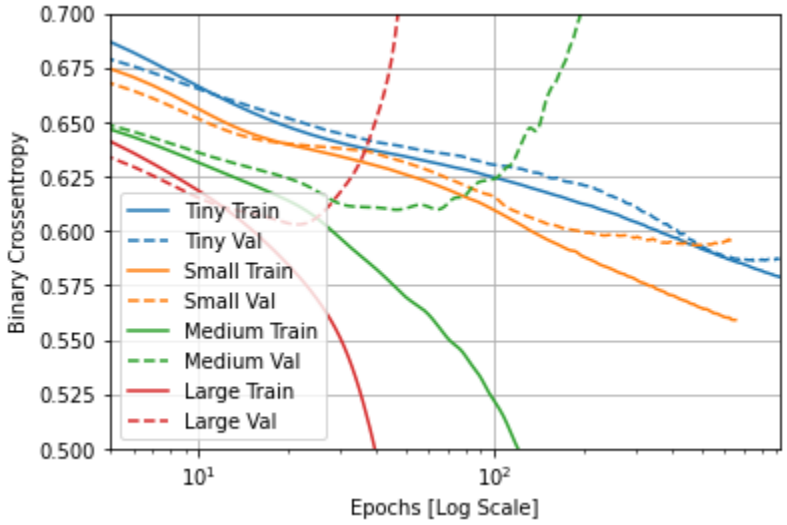
epoch 2 ⟶ mean squared error = 10.5479

weights = [0.2983, 0.3980, 0.4985, 0.5979, 0.7, 0.8]
(w_1 w_2 w_3 w_4 w_5 w_6)

زندگی یک انتخاب است و من شادی و خوشبختی را انتخاب می کنم.

۳. پاسخ سوال سوم

الف) پس از لودکردن دیتاست و انجام preprocessing روی دیتا و آماده کردنش برای آموزش مدل، توابعی برای تعریف کردن، compile و fit کردن مدل مینویسد. بعد با سه مدل مختلف که یکی کوچک، یکی متوسط و دیگری بزرگ است (اندازه مدل بر اساس تعداد لایه های آن و تعداد unit در هر لایه تعریف میشود، در واقع "ظرفیت" مدل است که تعداد پارامترهای مورد آموزش مدل آن را مشخص میکند) شروع میکند. خلاصه نتایج در شکل زیر آورده است. البته جزییات دقت ها و ضررها در epoch های مختلف هر مدل هم آورده است.

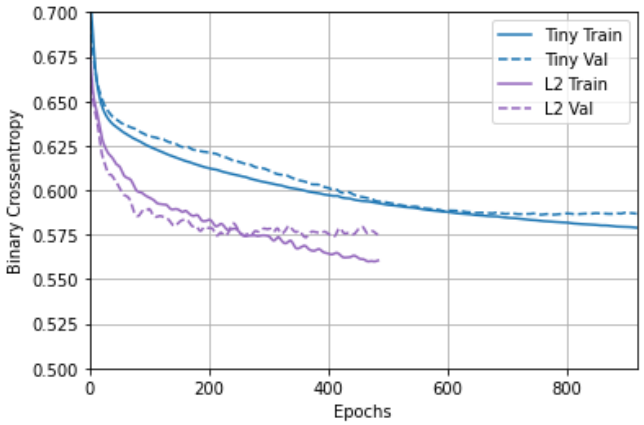


تمام این مدل ها با استفاده از کالبک EarlyStopping در جایی که مدل دیگر پیشرفت خاصی نمیکرده متوقف شده اند. مشخص است که به جز مدل tiny که فقط یک لایه مخفی با ۱۶ نورون دارد، همه مدل ها تا حدی overfit شده‌اند. و هر چقدر مدل بزرگتر بوده زودتر overfit کرده است. یعنی loss روی داده آموزشی تا حد خوبی کاهش یافته اما روی داده ارزیابی به طرز معناداری بیشتر است. حتی مدل small هم با زیاد شدن epoch به نظر میرسد در حال overfit است. برای رفع این مشکل از روش های مختلفی استفاده کرده است:

- اضافه کردن weight regularization

به دو روش l1 (نرمال کردن وزن ها بر اساس قدر مطلق آن ها) و l2 (نرمال کردن وزن ها بر اساس مربع آن ها) این کار را میتوانیم انجام دهیم. که l1 وزن ها را به سمت صفر میل میدهد و درواقع بردار وزن ها را sparse میکند ولی l2 وزن ها را جریمه میکند و کوچک میکند و معمولا موثرتر است. به همین خاطر اینجا از l2 استفاده

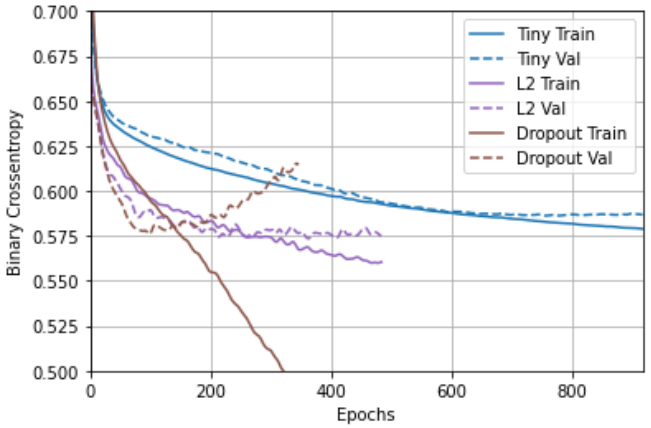
کرده و همان مدل large قسمت قبل را با آن پیاده سازی کرده است (اسم مدل را L2 گذاشته است) که نتیجه به این صورت شده :



تا حد خوبی مدل نسبت به overfit شدن بهینه شده است و به چیزی که انتظار داشتیم یعنی loss کمتر در مدل پیچیده تر نسبت به مدل tiny روی داده ارزیابی رسیده ایم. که بدون weight regularization نتوانسته بودیم. البته باز هم در اواخر آموزش مدل به سمت overfit شدن رفته است.

- اضافه کردن dropout

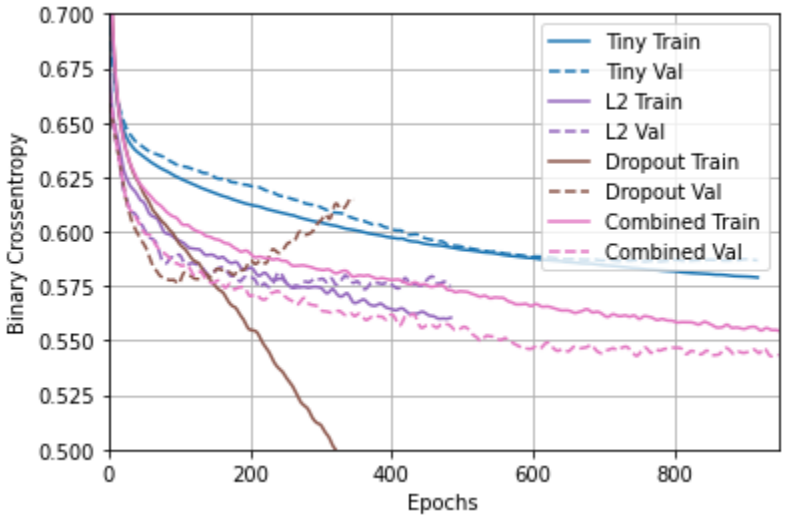
به هر کدام از لایه های همان مدل large (که ۴ لایه مخفی هرکدام با ۵۱۲ نورون داشت)، dropout با احتمال ۰.۵ اضافه میکند. تا هر بار در هنگام آموزش مدل نیمی از وزن ها صفر در نظر گرفته شوند و به نوعی همه نورون ها به تغییرات ورودی و نورون های دیگر مقاوم شوند. این کار برای جلوگیری از overfit شدن مناسب است همانطور که با رنگ قهوه ای خروجی این مدل آورده شده است:



در مقایسه با حالتی که فقط مدل large بود عملکرد بهتر است اما هنوز مانند tiny نیست که overfit نداشته باشیم.

- ترکیب l2 و dropout

حال هر دوی موارد گفته شده را اعمال کرده و نتایج با رنگ صورتی مشخص شده است:



که میبینیم اصلا شبکه overfit نکرده و مقدار ضرر روی داده ارزیابی حتی از آموزشی کمتر هم شده و از مدل tiny هم بسیار کمتر است که این استفاده مناسب از شبکه عمیق تر را نشان میدهد.

پس نتیجه میگیریم که راه های جلوگیری از overfit شدن شبکه: کوچکتر کردن شبکه تا جایی که به دقت لطمه نزند (چون دیدیم که برای tiny مشکل نداشتیم)، اضافه کردن weight regularization و اضافه کردن dropout بود. همچنین مواردی مانند batch normalization، data augmentation و داشتن داده های آموزشی بیشتر هم میتواند موثر باشد. که در این مطلب بررسی نشده اند. و اعمال چندتایی این موارد با یکدیگر میتواند کاملا مدل را از overfit نجات دهد.

ب) مدل های مختلف به ترتیبی که استفاده کردم و تغییر دادم در فایل HW3.ipynb آمده است. (لینک گوگل کولب) در نهایت بهترین مدل به دقت ۶۸ روی داده آموزشی و ۷۰ روی داده ارزیابی رسید. که با توجه به دقت خوب روی داده آموزشی underfit نکرده و با توجه به دقت معقول روی داده تست overfit هم نشده است. دو تا از بهترین مدل ها در این بخش:

```
model = tf.keras.Sequential([
    layers.Dense(512, activation='elu', input_shape=(FEATURES,), kernel_regula
rizer=regularizers.L1L2(0.0001)),
    layers.Dropout(0.3),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L1L2(0
.0001)),
    layers.Dropout(0.3),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L1L2(0
.0001)),
    layers.Dropout(0.3),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L1L2(0
.0001)),
    layers.Dropout(0.3),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L1L2(0
.0001)),
    layers.Dropout(0.3),
    layers.Dense(1)
])
```

و بهترین مدل ما:

```
model = tf.keras.Sequential([
    layers.Dense(512, activation='elu', input_shape=(FEATURES,), kernel_regula
rizer=regularizers.L2(0.0001)),
    layers.Dropout(0.5),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L2(0.0
001)),
    layers.Dropout(0.5),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L2(0.0
001)),
    layers.Dropout(0.5),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L2(0.0
001)),
    layers.Dropout(0.5),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L2(0.0
001)),
    layers.Dropout(0.5),
```

```
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L2(0.0
001)),
    layers.Dropout(0.5),
    layers.Dense(512, activation='elu', kernel_regularizer=regularizers.L2(0.0
001)),
    layers.Dropout(0.5),
    layers.Dense(1)
])
```