
**دانشکده مهندسی کامپیوتر**

**دکتر محمدی**

**بهار ۱۴۰۰**

---

# تمرین هشتم

# یادگیری عمیق

**مجتبی نافذ 96431335**

---

۱. به طور خلاصه شباهت‌ها و تفاوت‌های شبکه‌های عصبی بازگشتی (RNN) و همگشتی (CNN) را بنویسید (برای راحتی می‌توانید از کتاب یا اسلایدها و یا این لینک استفاده کنید)

**CNN:**

**مناسب برای داده های پراکنده مانند تصویر**

**در کل CNN ها ابزار قدرتمند تری نسبت به RNN ها به شمار می روند.**

**طول ورودی و خروجی ثابت است و تغییری نمیکند.**

**CNN ها نوعی از شبکه های عصبی، feed forward هستند که کمترین پیش پردازش را نیاز دارند.**

**در واقع CNN ها برای پردازش ویدیو و تصویر خاص منظوره هستند.**

**در CNN ها سلولهای عصبی، جداگانه به گونه ای سازمان یافته اند، که به صورت مناطقی همپوشان به ورودی پاسخگویی می کنند.**

**RNN:**

**مناسب برای داده های sequential و ترتیبی**

**تعداد feature های کم تر و قابلیت , توانایی کمتر نسبت به CNN**

**مدل RNN یک مدل دارای loop است برای مدیریت داده های sequential حافظه نیاز دارد. که در این loop از حافظه استفاده می کند.**

**به صورت اصلی RNN ها بر روی اطلاعات time series گذشته موثر کار میکند.**

**مدل RNN بیشتر در حوزه speech analysis و text analysis کاربرد دارد.**

۲. تعداد پارامترهای شبکه زیر را محاسبه کنید. (مراحل محاسبات خود را یادداشت کنید)

```
model = Sequential()
model.add(Embedding(max_features=1000, 8, input_length=max_len))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(64, return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

```
model.add(keras.layers.Embedding(1000, 8, input_length=10))
```

**Parameter's = (number of word in problem space state) * (number of coding variable)**

**= 1000 * 8 = 8000**

```
model.add(keras.layers.SimpleRNN(32, return_sequences=True))
```

**Parameter's = ( number of output |) * ( number of output + number of input + bias )**

**= 32 * ( 32 + 8 + 1 ) = 32 * 41 = 1312**

```
model.add(keras.layers.SimpleRNN(64, return_sequences=False))
```

**Parameter's = ( number of output |) * ( number of output + number of input + bias )**

**= 64 * ( 64 + 32 + 1 ) = 6208**

```
model.add(keras.layers.Dropout(0.5))
```

**No parameter**

```
model.add(keras.layers.Dense(10, activation='softmax'))
```

**Parameter's = (number of Dense layer neuron)*(number of previous layer output + bias )**

**= 10 * (64 + 1) = 650**

**دقت کنید dropout تاثیری در پارامتر های لایه ی آخر نداشت چون این پارامتر ها در برخی epoch صفر می شوند و گاها هم تاثیر گذار اند و در کل همه ی آن ها وجود دارند.**

۳. در این تمرین می‌خواهیم شبیه‌سازی بر روی مجموعه داده Newsgroup20 انجام دهیم. در این تمرین آزمایش‌های زیر را در این نوت‌بوک انجام دهید و نتایجی که به دست می‌آورید را مقایسه کنید و مناسب بودن یا نبودن هر روش را بررسی و تحلیل کنید که به کار بردن کدام روش نسبت به دیگری مناسب‌تر بوده است. (در مورد لایه‌های آموزشی استفاده از یک لایه GRU و یک لایه Dense کافی است و در صورت نیاز از می‌توانید از روش‌های Regularization استفاده کنید. نتایج تست مدل‌هایتان را که می‌خواهید در مقایسه تحلیل کنید در نوتبوک نگه دارید و حذف نکنید).

الف) هر کاراکتر را توسط یک عدد صحیح مشخص کنید (به عنوان مثال کد کلمه the برابر با عدد صحیح ۲ است). در این حالت ورودی مدل دنباله‌ای از اعداد خواهد بود. در این قسمت لازم است ابتدا هر عدد به صورت یک بردار دارای طول ۱ تبدیل شود که دستور `np.expand_dims` این کار را انجام می‌دهد.

ب) از فرمت one-hot برای هر کاراکتر استفاده کنید (در این حالت کد مربوط به هر کلمه یک بردار خواهد بود که فقط یک مقدار آن که متناسب با کد صحیح آن است ۱ خواهد بود). از آنجائیکه تعداد کلمه‌ها زیاد است، تبدیل کردن تمام داده‌های آموزشی به این فرمت نیاز به حافظه زیادی دارد. به همین دلیل، در ابتدای شبکه یک لایه با استفاده از لایه پرکاربرد Lambda تعریف می‌کنیم که عدد صحیح را به فرمت one-hot تبدیل کند.

پ) از لایه Embedding غیر قابل آموزش با وزن‌های تصادفی استفاده کنید. هدف از این بخش مقایسه این نوع کدگذاری کلمات که قابل آموزش نیست با دو حالت قبل است (برای freeze کردن این لایه کافی است trainable=False قرار دهید).

پ) از لایه Embedding قابل آموزش با وزن‌های تصادفی استفاده کنید (trainable=True قرار دهید).

ت) از لایه Embedding غیر قابل آموزش با وزن‌های پیش‌آموخته استفاده کنید. در کدی که در اختیار دارید، وزن‌های پیش‌آموخته دانلود و در متغیر embedding_matrix قرار داده شده‌اند. برای استفاده از آنها کافی است در لایه Embedding از weights=[embedding_matrix] استفاده کنید).

ث) از لایه Embedding قابل آموزش با وزن‌های پیش‌آموخته استفاده کنید.

**الف)**

**در مورد اول که لغت ها را با integer کد کنیم. ارتباط معنایی بین کلمات که کلا نادیده گرفته می شود و همچنین زمانی که داریم جریمه می کنیم این جریمه عادلانه نیست و مفهوم درستی هم ندارد چون برای نمونه :**

**تلاش کد: ۲۳ ---- ایران کد: ۲۴ جریمه به نسبت اختلاف آن است که برابر ۱ است.**

**تلاش کد: ۲۳ ---- کوشش کد: ۹۵**

**<== جریمه از حالت قبلی خیلی بیشتر است در حالی که کلمات مترادف هستند**

**نتیجه آن که دقت حدود ۱۵ درصد در داده های تست گرفتیم که افتضاح است و در داده های آموزشی هم دقت حدود ۴۵ درصد گرفتیم که خوب نیست.**

ب)

در قسمت ب ما خیلی بهتر نتیجه گرفتیم

البته در آزمایش من overfit شد و تا 67 بیشتر نرفت

اما به راحتی می توانستیم با regularization جلوی overfit را بگیریم و خیلی بهتر از این هم نتیجه بگیریم.

در استفاده از One Hot نکته مثبت ما نسبت به integer گرفتن این است که مقدار خطا بین هیچ دو لغتی فرق ندارد و همگی فقط در یک بیت اختلاف دارند

اما همین خودش بزرگترین مشکل هست چرا که اصلا مترادف یا متضاد بودن کلمات در نظر گرفته نمیشود. و خطای در هر اشتباهی یکسان است.

پ)

در قسمت های بعدی از لایه ی embedding استفاده می شود که تمام ارتباط معنایی را هم در نظر می گیرد.

در این قسمت که وزن های embedding رندوم هستند و آپدیت نمی شوند. تا ۲۳ درصد دقت هم دست پیدا می کنیم و روی داده ها overfit هم نکردیم.

پ)

زمانی که وزن ها را قابل آموزش کردیم در نتیجه این شد که تا ۴۳ درصد دقت در تست رسیدیم و روی آموزشی overfit کردیم و تا ۸۹ درصد هم رسیدیم و نتیجه این که دقت اگر جلوی overfit را با regularization یا dropout بگیریم تا حد خوبی قابل افزایش است.

ت) در این حالت ما با سرعت بالایی به دقت ۶۷ درصد رسیدیم و روی آموزشی overfit کردیم و در همان ۶۷ درصد گیر کردیم.

ث)

در این حالت ما تا ۷۲ درصد دقت هم رسیدیم و در روی آموزشی overfit کردیم و متوقف شدیم. بهترین روش این گزینه بود و قابلیت بهبود بالایی هم داشت.