دانشکده مهندسی برق و کامپیوتر

استاد درس:

دستورکار آزمایشگاه هوش محاسباتی
جلسه

اهداف این جلسه

- ساخت یک دسته‌بندِ حالات با استفاده از TF Keras Sequential API
- ساختِ یک ConvNet برای شناسایی اعداد زبان اشاره با استفاده از TF Keras Functional API

در پایانِ این جلسه شما قادر خواهید بود که :

- برای دسته‌بندیِ دوتایی، یک ConvNet را ساخته و آموزش دهید.
- برای دسته بندیِ چند کلاسه، یک ConvNet را ساخته و آموزش دهید.
- تفاوت موارد استفاده از Squential و API Functional ها را شرح دهید.

۱ آماده‌سازی

در ابتدا، همانطور که در فایل ژوپیتر مشاهده خواهید کرد، باید پکیج های مربوطه را فراخوانی کنیم . شما در این تمرین از مجموعه داده‌ی `Happy House` استفاده خواهید کرد. این مجموعه داده حاوی تصاویرِ RGB با اندازه‌ی 64x64 پیکسل از صورت های افراد است. وظیفه ی شما ساختن یک شبکه ی پیچشی است که تشخیص دهد آیا صورت ها در حال لبخند زدن هستند، یا خیر.

۲ لایه ها در TF Keras

در جلسه ی قبل، شما لایه ها را بصورت دستی با numpy پیاده‌سازی کردید. در Keras شما نیازی به نوشتن کد به صورت مستقیم برای این کار ندارید و Keras بصورت از پیش تعریف شده، لایه هایی را دارد که می‌توانید از آن‌ها استفاده کنید.

۳ Sequential API

در جلسه ی قبل، شما برای درکِ ساختارِ شبکه ی عصبیِ پیچشی ، توابع کمکی ای را با استفاده از `numpy` ساختید. بسیاری از اپلیکیشن های مبتی بر یادگیری عمیق ، با استفاده از چهارچوب [۱] های برنامه نویسی ای ساخته شده که درون خود، تعداد زیادی تابعِ از پیش نوشته شده دارند که به راحتی قابل فراخوانی هستند. Keras یک چهارچوب ساخته شده بر روی TensorFlow است که برای ما امکانِ استفاده‌ی راحت‌تر و بهینه تر از مفاهیم ساخت مدل و آموزش آن را فراهم میسازد.
برای قسمت اولِ این تمرین، شما یک مدل را با استفاده از Keras' Sequntial API خواهید ساخت. با این کار شما قادر خواهید بود که مدل خود را بصورت لایه به لایه بسازید. ساخت مدل بصورت لایه به لایه برای کابردهایی که مدل شما دقیقا یک تنسور ورودی و یک تنسور خروجی دارد، بسیار ایده آل خواهد بود.
همانطور که در ادامه خواهید دید، استفاده از Sequential API ساده و سرراست است، اما فقط برای وظایف ساده، کارایی دارد. در ادامه‌ی این تمرین ها، شما با استفاده از Functional API جایگزینی انعطاف پذیرتر و قدرتمندتر خواهید ساخت.

۱.۳ ساختِ مدلِ ترتیبی

همانطور که اشاره کردیم، TensorFlow Keras Sequential API می‌تواند برای ساخت مدل های ساده که بصورت ترتیبی پیش می‌روند، استفاده شود.
شما همچنین می‌توانید با استفاده از تابعِ `.add()` لایه هایی را به مدل خود اضافه کنید یا با استفاده از `.pop()` آن‌ها را حذف کنید. درست شبیه به لیست های معمولی در پایتون. درواقع مدلِ ترتیبی بسیار شبیه به لیست های پایتون است. اگر مدل شما غیر خطی است یا شامل لایه‌هایی با چندین ورودی و خروجی است، استفاده از مدل ترتیبی، گزینه‌ی خوبی نیست.
برای ساخت هر لایه در Keras ، شما باید شکل ورودی را مشخص کنید. این کار به این دلیل است که در Keras، شکل وزن ها بستگی به شکل ورودی ها دارد. وزن ها زمانی ساخته می‌شوند که مدل ابتدا تعدادی داده‌ی ورودی را مشاهده کند. مدل های ترتیبی می‌توانند با استفاده از فرستادن لیستی از لایه ها به سازنده‌ی ترتیبی [۲] ، ساخته شوند. موضوع تمرین بعدی، همین است.

[۱] framework
[۲] Sequential constructor

تمرین اول: happyModel

در فایل ژوپیتر این جلسه، در قسمتِ Exercise 1 - happyModel تابعِ `happyModel` را به منظور ساخت مدل زیر، تکمیل کنید:
`ZEROPAD2D -> CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> FLATTEN -> DENSE`
برای راهنمایی می‌توانید از این tf.keras.layers
همچنین پارامتر های زیر را برای همه مراحل، به کار ببرید:

- ZeroPadding۲D: لایه‌گذاری [۳] = ۳ ، شکل ورودی: ۶۴x۶۴x۳
- Conv۲D : از ۳۲ فیلتر ۷x۷ با گام [۴] ۱ استفاده کنید.
- نرمالسازی دسته ای [۵] : برای محورِ ۳
- ReLU
- MaxPool۲D : با استفاده از پارامتر های پیش‌فرض
- Flatten : خروجی قبلی
- لایه ی Dense Fully-connected: یک لایه ی fully connected با یک نورون و فعالسازِ سیگموید

راهنمایی: برای مختصرنویسی می‌توانید از tfl بجای $tensorflow.keras.layers$ استفاده کنید.
بعد از ساخته شدن مدلتان، می‌توانید آن را با استفاده از یک بهینه‌ساز و تابع هزینه‌ی دلخواه، برای آموزش، کامپایل کنید. هنگامی که رشته‌ی `accuracy` به عنوان یک سنجه، انتخاب شود، نوع آن، بر اساس نحوه استفاده‌ی تابع هزینه، به طور خودکار، تبدیل خواهد شد. این عمل، یکی از انواع بهینه‌سازی‌هایی است که در TensorFlow بصورت آماده، طراحی شده است. برای اطلاعات بیشتر، می‌توانید این لینک را مشاهده کنید سپس برای ارزیابی پارامتر های مدل خود، می‌توانید از `.summary()` استفاده کنید. با این کار، شما قادر خواهید بود که نوع لایه هایی که دارید، شکل خروجی ها و تعداد پارامتر های هر لایه را مشاهده کنید. خروجی، چیزی شبیه ذیل خواهد شد:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_1 (ZeroPaddin (None, 70, 70, 3)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 32)        4736
_________________________________________________________________
batch_normalization (BatchNo (None, 64, 64, 32)        128
_________________________________________________________________
re_lu (ReLU)                 (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 32769
=================================================================
Total params: 37,633
Trainable params: 37,569
Non-trainable params: 64
_________________________________________________________________
```

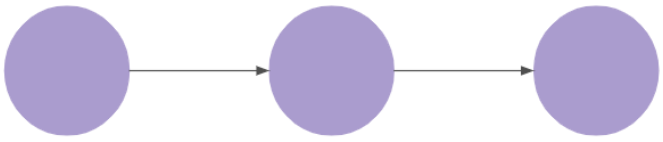
۲.۳ آموزش و ارزیابیِ مدل

بعد از ساختن و کامپایل کردن مدل با هزینه و بهینه‌سازِ دلخواه، می‌توانیم مدل خود را آموزش دهیم و آن را ارزیابی کنیم. برای آموزش مدل، کافیست از `.fit()` استفاده کنید. شما این اختیار را دارید که شماره‌ی epoch یا اندازه‌ی minibatch را تغییر دهید. (به طور مثال برای یک مجموعه داده‌ی unbatched) سپس می‌توانید با استفاده از `.evaluate()` مدل خود را در مقابل مجموعه ی آزمون، ارزیابی کنید.
این تابع مقدار تابع هزینه و پارامترهای معیاری که در حین کامپایل شدن، مشخص شده اند را چاپ خواهد کرد که در مثالِ ما، مقادیر آن به ترتیب، `binary_crossentropy` و `accuracy` خواهد بود.

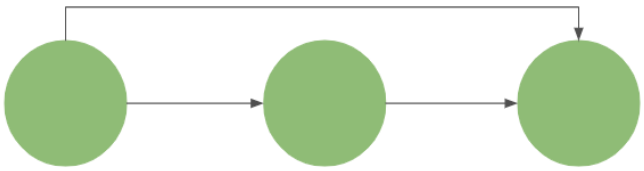
[۳] padding
[۴] stride
[۵] BatchNormalization

۴ Functional API

در این قسمت قصد داریم با استفاده از Functional API یک شبکه ی پیچشی را برای تمییز دادنِ ۶ شماره در زبان اشاره، بسازیم.
Functional API قابلیت اداره کردنِ مدلهایی با توپولوژیِ غیرخطی و لایه‌های اشتراکی مانند لایه‌هایی با چندین ورودی و خروجی را داراست.
اگر مدل ترتیبی را بصورت یک خط در نظر بگیریم، آنگاه می‌توان مدل تابعی را بصورت یک گراف در نظر گرفت که گره های آن می‌توانند به روش های زیادی به همدیگر متصل شوند. در مثالِ تصویریِ زیر، یک جهت احتمالی مدل ترتیبی، حرکت در مقابل اتصال skip نشان داده شده است ، که تنها یکی از روش های متعددی است که می توان یک مدل Functional ایجاد کرد . اتصالِ skip، همانطور که احتمالا حدس زده باشید، تعدادی از لایه ها را رد می‌کند و خروجی را توسط لایه‌های بعدی، تغذیه می‌کند.



Sequential Movement



Skip Connection

۱.۴ بارگزاریِ مجموعه داده‌ی SIGNS

مجموعه داده‌ی SIGNS حاویِ مجموعه ای از ۶ نشانه از اعداد ۰ تا ۵ می‌باشد. کد این قسمت در فایل ژوپیترِ جلسه، در بخشِ ۱.۴ پیاده‌سازی شده است و در قسمت بعدی کد، یک نمونه از تصویر برچسب‌گذاری‌شده را خواهید دید.
تصویر ۲

۲.۴ تقسیمِ داده ها به مجموعه ی آموزشی و آزمون

در فایل این جلسه، در قسمت ۲.۴ ، کد مربوط به تقسیمِ داده‌ها به دو قسمتِ آموزشی و آزمون را مشاهده می‌کنید. از آنجایی که مجموعه داده‌ی ما، شامل داده‌های تصویری است، طبیعی است که یک ConvNet بر روی آن اعمال کنیم.

۳.۴ انتشارِ پیش‌رو

در TensorFlow توابع از پیش ساخته شده ای وجود دارند که مراحل پیچش را برای شما پیاده سازی خواهند کرد. تا اینجا شما باید با نحوه‌ی ساخته شدن گراف های محاسباتی توسط TensorFlow آشنا شده باشید. در Functional API شما گرافی از لایه ها می‌سازید و این کار به مدل شما انعطاف زیادی می‌دهد.
اگرچه، مدلی که در ادامه خواهیم دید، می‌تواند با استفاده از مدل ترتیبی نیز تعریف شود زیرا جریان اطلاعات در این مورد به صورت خطی است. اما برای یادگیری، از Functional API استفاده خواهیم کرد.

برای ساخت گراف خود، از ساخت یک گره ی ورودی که به صورت یک تابع قابل فراخوانی عمل می کند، کار خود را آغاز کنید.

- **input_img = tf.keras.Input(shape=input_shape)**

سپس با فراخوانیِ یک لایه بر روی شیءِ `input_img` ، یک گره جدید بر روی گراف لایه ها ایجاد کنید.

- **tf.keras.layers.Conv2D(filters= ... , kernel_size= ... , padding='same')(input_img)** :

توضیحات بیشتر در Conv۲D

- **tf.keras.layers.MaxPool2D(pool_size=(f, f), strides=(s, s), padding='same')**

  ورودی خود را با استفاده از یک پنجره با اندازه (f،f)

  و گام های به اندازه (s،s) برای انجام max pooling بر روی هر پنجره ، نمونه برداری می کند.

- **tf.keras.layers.ReLU()**:

  مقدار نظیر به نظیرِ درایه ایِ Z ReLU of (که می‌تواند هر شکلی داشته باشد) را محاسبه می‌کند. توضیحات بیشتر در ReLU

- **tf.keras.layers.Flatten()**:

  با داشتنِ تنسورِ P این تابع به صورت دسته ای، هر نمونه ی آموزشی یا آزمون را می‌گیرد و در خروجی یک بردار یک بُعدی به ما می دهد. توضیحات بیشتر در Flatten

- **tf.keras.layers.Dense(units= ... , activation='softmax')(F)**:

  با داشتنِ ورودیِ مسطح شده‌ی F ، خروجی ای را که با استفاده از یک لایه ی تمام متصل، محاسبه شده است، برمیگرداند. توضیحات بیشتر در Dense

در آخرین تابعی که در بالا آمده است، لایه ی fully connected بصورت خودکار، وزن ها را در گراف مقداردهی اولیه میکند و سپس به آموزش آنها در حین آموزش مدل، ادامه میدهد. از این رو، شما به مقداردهی اولیه به آن وزنها در هنگام مقداردهی اولیه به پارامترها، نیاز پیدا نکردید.

در نهایت ، قبل از ایجاد مدل ، باید خروجی را با استفاده از آخرین ترکیب تابع تعریف کنید (در اینجا، لایه ی Dense) :

- **outputs = tf.keras.layers.Dense(units=6, activation='softmax')(F)**

پنجره،کرنل،فیلتر،تجمیع

کلماتِ *کرنل* و *فیلتر*هر دو به یک موضوع اشاره می‌کنند. کلمه‌ی *فیلتر*تعداد *کرنل* هایی که در یک لایه‌ی پیچشی استفاده خواهند شد را، مشخص می‌سازد *تجمیع* اسمِ عملیاتی است که بیشینه یا متوسطِ کرنل‌ها را می‌گیرد.

به همین دلیل است که پارامترِ `pool_size` به `kernel_size` اشاره دارد و شما از `(f,f)` برای اشاره به اندازه فیلتر استفاده می‌کنید.

اندازه تجمیع و اندازه کرنل هر رو به یک چیز ولی در موضوعات مختلف - به شکل پنجره‌ای که عملیات در آن در حال انجام است - اشاره می‌کنند.

تمرین دوم: convolutional_mode

در فایل ژوپیتر این جلسه، در قسمت Exercise 2-convolutional_model تابعِ `convolutional_model` زیر را با توجه به مدلِ `CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN -> DENSE` تکمیل کنید. از توابعی که در بالا به آنها اشاره کردیم، استفاده کنید. در حین این کار، از پارامتر های زیر استفاده کنید:

- Conv2D: 8 4x4 filters, stride 1, padding=SAME
- ReLU
- MaxPool2D: 8x8 filter size, 8x8 stride, padding=SAME
- Conv2D: 16 2x2 filters, stride 1, padding=SAME
- ReLU
- MaxPool2D: 4x4 filter size, 4x4 stride, padding=SAME
- Flatten the previous output
- Fully-connected (Dense) layer: یک لایه ی تمام متصل با ۶ نورون و فعالسازِ سافت-مکس را اعمال کنید.

هر دو روشِ ترتیبی یا تابعی، یک شیء از نوع TF Keras mode برمی‌گردانند. تنها تفاوت این دو این است که ورودی ها چگونه در آنها مدیریت می‌شوند.

۴.۴ آموزشِ مدل

در فایل این جلسه در کد قسمتِ ۴.۴ شما مدل خود را آموزش می‌دهید.

۵ History Object

شیءِ history یکی از خروجی های `.fit()` است و اطلاعات هزینه ها و مقادیر متریک را در حافظه نگهداری می‌کند. این مقدار به صورت یک واژه‌نامه ذخیره شده است که میتوانید آن را توسط `history.history` بازیابی کنید و سپس مقدار هزینه را نسبت به زمان با استفاده از آن نمایش دهید