



دانشگاه تهران
پردیس فارابی
دانشکده مهندسی
گروه مهندسی کامپیوتر

پیاده سازی برنامه دسکتاپ با رابط گرافیکی برای سیستم تشخیص کلمات کلیدی
فارسی در صوت

نگارش:

سیدسینا موسوی ثابت

استاد راهنما:

دکتر زهرا موحدی

گزارش پروژه برای دریافت درجه ی کارشناسی در رشته مهندسی کامپیوتر

شهریور ۱۴۰۱

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

چکیده

دستیارهای صوتی نرم‌افزار هوشمندی هستند که به دستورات صوتی پاسخ می‌دهند و می‌توانند بر روی هر دستگاهی از جمله تلفن‌های هوشمند، بلندگوها، رایانه‌های رومیزی/لپ‌تاپ، تبلت‌ها، ابزارهای پوشیدنی، کنسول‌های بازی، کنسول‌های تلویزیون، هدست‌های واقعیت مجازی (VR)، اتومبیل‌ها و دستگاه‌های اینترنت اشیا اجرا شوند. دستگاه‌های (IoT) به عنوان مثال می‌توان به الکسای آمازون، سیری اپل، دستیار گوگل و کورتانای مایکروسافت اشاره کرد.

در این پروژه، با استفاده از دیتاست‌های موجود در اینترنت و جمع‌آوری دیتا و استخراج ویژگی‌های مورد نیاز از سیگنال‌های صوتی و استفاده از مدل‌های یادگیری عمیق CNN و RNN، مدلی برای تشخیص کلمات کلیدی در صوت را ایجاد کردیم که قادر به تشخیص کلمات و مشخص کردن محل رخ داد آن در صوت می‌باشد. هدف از انجام این پروژه پیاده‌سازی یک دستیار صوتی زبان فارسی با استفاده از منابع موجود در اینترنت بود.

که در نتیجه کار یک برنامه ی دسکتاپ برای نمایش گرافیکی صوت و رابط کاربری گرافیکی پیاده سازی شد.

کلمات کلیدی: بازشناسی الگو، پردازش گفتار، یادگیری عمیق، سیگنال صوتی، برنامه دسکتاپ، دستیار صوتی

۱ فصل اول

۱-۱ مقدمه..... ۷

۲ فصل دوم

۲-۱ مروری بر کارهای مشابه ۹

۳ فصل سوم

۳-۱ شرح کلی سیستم ۱۱

۳-۲ بررسی الگوریتم های یادگیری عمیق ۱۲

۳-۲-۱ شبکه های عصبی مصنوعی ۱۲

۳-۲-۲ شبکه های عصبی کانوولوشنال ۱۸

۳-۲-۳ شبکه های عصبی بازگشتی ۲۲

۳-۳ معیارهای سنجش عملکرد مدل های یادگیری ماشین..... ۲۸

۳-۴ ابزار ها ۲۹

۴ فصل چهارم

۴-۱ چند نخی..... ۳۰

۴-۲ شروع یک نخ جدید ۳۰

۴-۳ مازول Threading ۳۲

۴-۴ توضیح کد app.py..... ۳۳

۴-۵ توضیح کد real_time.py..... ۴۷

۴-۶ مزایای multi-threading در پایتون..... ۵۱

۵ فصل پنجم

۵-۱ ارائه معماری برای مدل ها ۵۲

۵-۱-۱ مدل RNN 52

- ۵-۲ نگاه‌ی به جهان یادگیری ماشین.....۵۲
- ۵-۳ ساختار شبکه‌های عصبی بازگشتی (RNN).....۵۳
- ۵-۴ توضیح کد `model.py`.....۵۶
- ۵-۵ راه‌حل‌های استفاده شده برای `overfitting`.....60
- ۵-۶ مقایسه عملکرد مدل‌ها.....۶۱

۶ فصل ششم

- ۶-۱ خلاصه.....۶۲
- ۶-۲ نتیجه‌گیری.....۶۲
- مراجع.....۶۳
- پیوست – نحوه کار با برنامه دسکتاپ.....۶۴

فصل اول

۱-۱ مقدمه

چگونه یک دستیار صوتی می تواند حرف ما را بفهمد؟ خوب، آنها بر اساس سیستم ASR (تشخیص خودکار گفتار) هستند. ابتدا این سیستم‌ها گفتار را ضبط می‌کنند و سپس آن را به صداها یا گروه‌هایی از آنها (واج) که به متن ترجمه می‌شوند، تجزیه می‌کنند. دستیار همچنین می تواند لحن، فرکانس و صداها جنبه دیگر از صدای یک فرد را تجزیه و تحلیل و «به خاطر بسپارد» تا دقیقاً بداند چه کسی با آن صحبت می کند.

دستیار صوتی زمانی شروع به کار می کند که کاربر آن را با یک کلمه بیدار فعال کند. کاربران آمازون الکسا این نام را برای فعال کردن دستیار صوتی می گویند، در حالی که کاربران اپل می توانند برای انجام همین کار، "Hey Siri" را بگویند.

بنابراین، برای مثال، می‌توانید از سیری در آیفون خود بپرسید: «سیری، امروز هوا چگونه است؟». هنگامی که دستگاه بیدار می شود و سؤال شما را به فرمان ماشینی ترجمه می کند، اطلاعات را از اینترنت یا یک برنامه کاربردی برای یافتن پاسخ بازیابی می کند - در این مورد، پیش بینی آب و هوای آن روز را به صورت آنلاین ارائه می دهد.

با این حال، استفاده از هوش مصنوعی و به ویژه یادگیری ماشینی است که به دستیارهای صوتی این امکان را می دهد که با گذشت زمان هوشمندتر شوند. به لطف این فناوری‌ها، برنامه‌های دستیار صوتی مدرن دیگر نیازی به تکیه بر واژگان از پیش نصب شده محدود ندارند، بلکه می‌توانند به میلیون‌ها کلمه و عبارت در فضای ابری دسترسی داشته باشند. به لطف الگوریتم‌های یادگیری ماشینی، دستیارهای صوتی همچنین می‌توانند به یک سخنرانی کامل گوش دهند تا متن را به جای هر کلمه به طور جداگانه درک کنند.

هدف این پروژه ایجاد سیستمی برای تشخیص کلمات کلیدی مشخصی در صوت می باشد که شامل گام های زیر برای انجام آن میشود:

- ۱- جمع آوری داده ها - جمع آوری داده های خام موجود در اینترنت و جمع آوری آنها به صورت دستی
- ۲- تبدیل داده های خام به داده های قابل استفاده - پاکسازی و استخراج بخش های مفید دیتاست ها و لیبیل زدن آنها
- ۳- پیش پردازش داده ها و ایجاد دیتاست - تبدیل داده ها به فرمت قابل استفاده برای مدل های یادگیری عمیق و جمع آوری آنها در یک دیتاست واحد برای سهولت در فاز train مدل

۱- مشخص کردن معماری مدل ها - مشخص کردن جزئیات مدل ها

۲- پیاده سازی مدل ها و train آنها

۳- بهینه سازی عملکرد مدلها

۴- استفاده از چند نخی برای کار با مدل در کنار برنامه اصلی

۵- پیاده سازی برنامه دسکتاپ

در ادامه به مرور کارهای مشابه و شرح گام های بالا خواهیم پرداخت.

فصل دوم

۱-۲ مروری بر کارهای مشابه

در آینده نزدیک، انتظار می رود دستیارهای صوتی نیز نقش فعال تری را ایفا کنند. دستیاران به جای اینکه فقط منتظر دستورات کاربر باشند، اطلاعات مربوط به زمینه را جمع آوری می کنند و سپس با ارائه پیشنهادات مفید به کاربر، ابتکار عمل را بر عهده می گیرند.

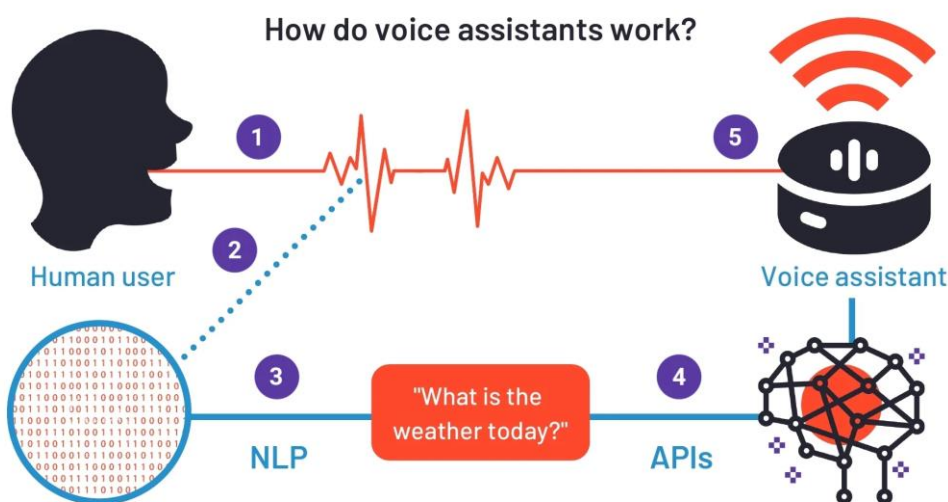
ردیف	شماره مرجع	محتویات پژوهش
۱	[1]	مروری بر روش ها و پیشرفت های یادگیری عمیق در زمینه ی پردازش صوت و بررسی تکنیک های به روز استفاده شده در دنیای واقعی
۲	[2]	ارائه روش بهینه برای سنتز صوت و تبدیل متن به صوت با استفاده از روش های یادگیری عمیق و مدل های شبکه عصبی بازگشتی
۳	[3]	ارائه مدلی برای تشخیص فعالیت صوتی و مکالمه در صوت و استخراج آن
۴	[4]	استفاده از یک سیستم برای تشخیص دو زبان متفاوت انگلیسی و چینی با استفاده از یادگیری عمیق
۵	[5]	تشخیص صدای گریه نوزاد توسط یادگیری عمیق
6	[6]	تشخیص هوشمند اختلال گفتاری با استفاده از سیستم پردازش سیگنال صوت

از دیگر موارد دیگر میتوان به دستیار های صوتی اشاره کرد. برای مثال دستیار صوتی شرکت آمازون که Alexa نام دارد و یا دستیار صوتی شرکت اپل که Siri نام دارد و یا دستیار صوتی گوگل که Google assistant نام دارد. این دستیارهای صوتی با توجه به دسترسی این شرکت ها به مقادیر حجیم دیتا عملکرد بسیار خوبی دارند و با داشتن یک مدل زبان انگلیسی، قابلیت پردازش بر روی گفتار را دارند و میتوانند دستورات کاربر را اجرا کنند و باعث سهولت استفاده تلفن های هوشمند میشوند. به زودی، دستیارهای صوتی همچنین میتوانند با شناسایی یک صدا و تطبیق آن با یک کارت اعتباری یا حساب بانکی، خریدها را تأیید کنند. کاربران می توانند به سادگی با استفاده از دستورات صوتی هزینه سفارشات خود را پرداخت کنند - دستیار صوتی فقط از آنها می خواهد که پرداخت را تأیید کنند. از دیگر موارد استفاده از پردازش صوت میتوان به دستیار های صوتی برای افراد نابینا اشاره کرد که با خواندن اطلاعات روی صفحه نمایش تلفن همراه یا سیستم های خانگی با لحنی نزدیک به انسان، امکان استفاده از این محصولات را به این افراد میدهند. همه ی تنها بخش اندکی از پیشرفت های هوش مصنوعی و دستیارهای صوتی را نمایان می کند.

فصل سوم

۳-۱ شرح کلی سیستم

در این قسمت به شرح کلی سیستم و توضیح الگوریتم های مورد استفاده میپردازیم. در سیستم پیاده سازی شده این پروژه، داده های ورودی از طریق میکروفون یا بصورت فایل های صوتی به سیستم داده میشود و سپس یک سری اقدامات بر روی این داده ی ورودی انجام میگردد تا این داده ها قابل استفاده توسط مدل های ما شوند. در این بخش ابتدا سیگنال صوتی به بخش هایی با اندازه مساوی تقسیم میشود (در این پروژه به قسمت های ۱ ثانیه ای). این تقسیم بندی به صورتی انجام میشود که قسمت ها با هم اشتراک داشته باشند تا اگر کلمه ی مورد نظر مدل در این صوت گفته شده بود در این قسمت بندی از بین نرود و برای مثال نیمی از آن در یک قسمت و باقی آن در قسمت دیگر نیفتد. جلوتر در رابطه با روش دقیق این کار توضیحات تکمیلی خواهیم آورد. بعد از این تقسیم بندی، ویژگی ها از این قسمت ها استخراج میشوند و آماده ی استفاده برای مدل ها میشوند. سپس این ویژگی های استخراج شده به عنوان ورودی به مدل داده میشود تا مدل پیشبینی را انجام دهد. پس از آن خروجی به کاربر به نمایش در آورده میشود. در انتها نیز به پیاده سازی این موارد در قالب یک اپ دسکتاپ خواهیم پرداخت.



تصویر شماره ۱

در فصل های بعدی به تفکیک و با جزئیات به فعالیت های انجام شده برای این پروژه خواهیم پرداخت. در ادامه کد های این پروژه و فصل مورد نظر آنها را مشاهده میکنید:

- کد `app.py` برای طراحی رابط کاربری گرافیکی. فصل ۴
- کد `real_time.py` گرفتن صوت از ورودی میکروفون در حین اجرا و رسم نمودار و سپس فرستادن یک ثانیه ی صوت به `model.py`. فصل ۴
- کد `model.py` برای ساخت مدل ها به منظور `train` کردن آنها. فصل ۵

۳-۲ بررسی الگوریتم های یادگیری عمیق

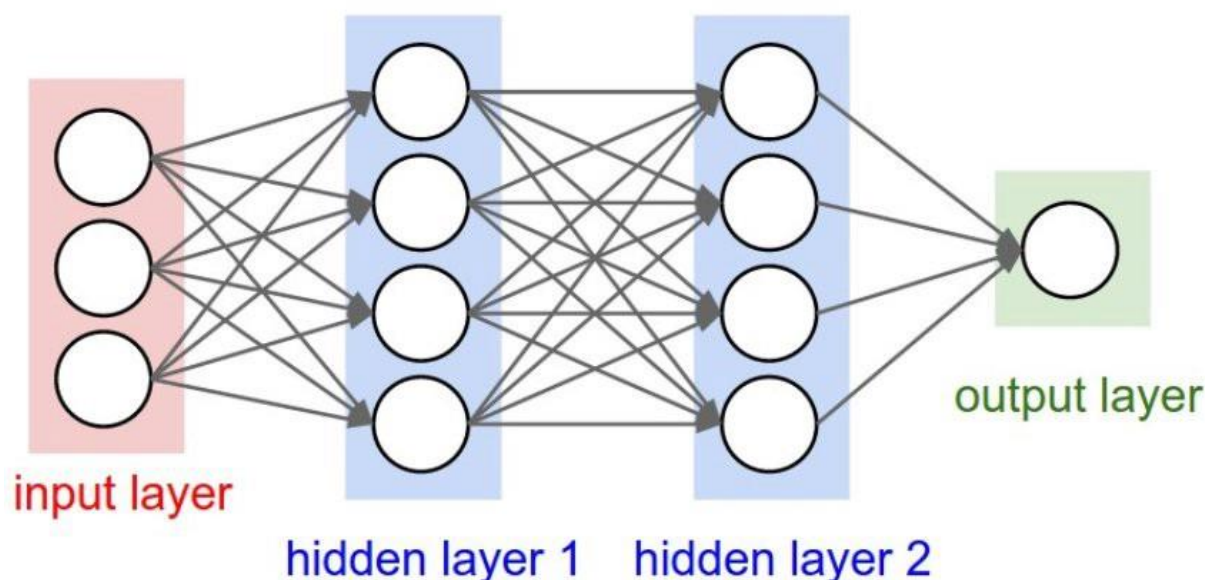
در این بخش به بررسی الگوریتم های مختلف یادگیری عمیق و استفاده شده در این پروژه خواهیم پرداخت.

در این پروژه از الگوریتم های یادگیری عمیق برای ایجاد مدل ها استفاده شده. از جمله این مدل ها شبکه عصبی کانولوشنال و شبکه عصبی بازگشتی هستند که در این پروژه مورد استفاده قرار گرفته اند.

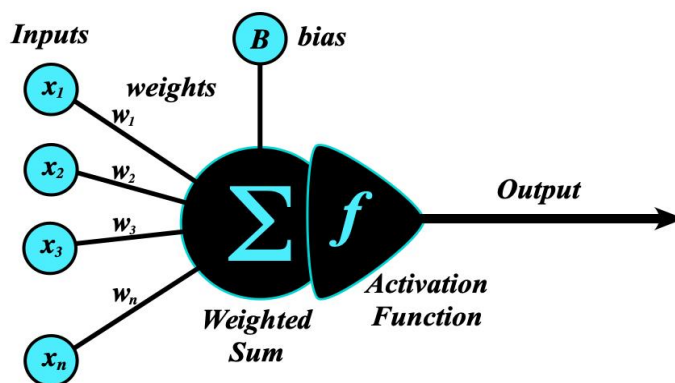
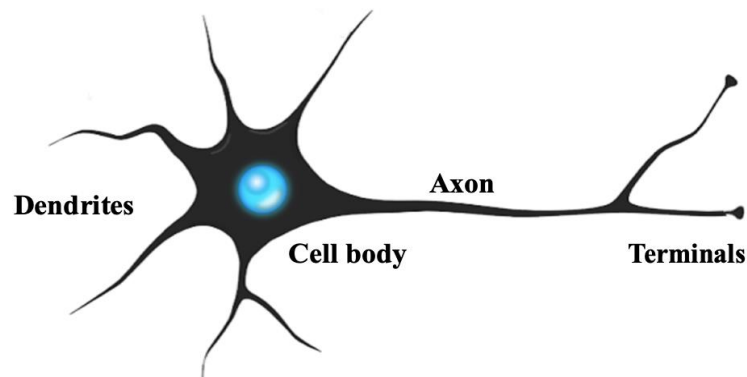
یادگیری عمیق یک زیرشاخه از یادگیری ماشین و بر مبنای مجموعه ای از الگوریتم ها است که در تلاشند تا مفاهیم انتزاعی سطح بالا در دادگان را مدل نمایند که این فرایند با استفاده از یک گراف عمیق که دارای چندین لایه پردازشی متشکل از چندین لایه تبدیلات خطی و غیرخطی هستند، مدل میکنند. انگیزه نخستین در به وجود آمدن این ساختار یادگیری از راه بررسی ساختار عصبی در مغز انسان الهام گرفته شده است که در آن یاخته های عصبی با فرستادن پیام به یکدیگر درک را امکان پذیر میکنند، بسته به فرض های گوناگون در مورد نحوه اتصال این یاخته های عصبی، مدل ها و ساختار های مختلفی در این حوزه پیشنهاد و بررسی شده اند، هرچند که این مدل ها به صورت طبیعی در مغز انسان وجود ندارد و مغز انسان پیچیدگی های بیشتری را داراست. از جمله این مدل ها میتوان به مدل های شبکه عصبی عمیق، شبکه عصبی کانولوشنال و شبکه عصبی بازگشتی اشاره کرد.

۳-۲-۱ شبکه های عصبی مصنوعی

یک شبکه عصبی یا شبکه عصبی عمیق (Artificial Neural Network) از یک لایه ورودی، یک لایه خروجی و چندین لایه ی پنهان میان این دو لایه تشکیل میشود. هر لایه از تعدادی نرون تشکیل شده است.



این نرون ها از بخش های مختلفی تشکیل شده است که تصویر شماره ۲ نابع فعال سازی غیرخطی و وزن ها بر روی اتصالات ورودی و خروجی آن در قالب یک تبدیل خطی میشود.



تصویر شماره ۳

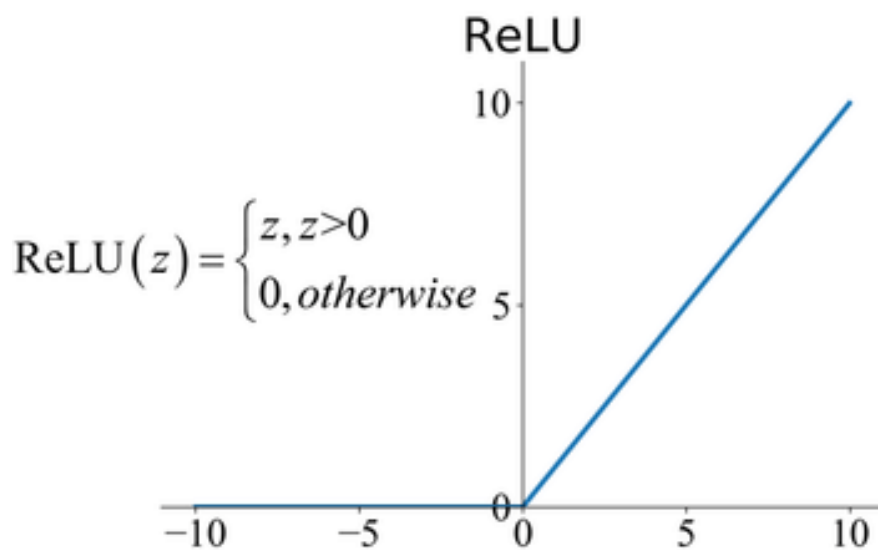
در این نرون یک بردار از وزن ها و یک مقدار bias نگهداری میشود که در نهایت پس از اعمال آنها روی ورودی های نرون، حاصل ها با یکدیگر جمع میشوند. این وزن ها اهمیت هر ورودی را نشان میدهند و تاثیر هر یک را روی خروجی این نرون مشخص میکنند. در پروسه ی آموزش شبکه عصبی مقادیر این وزن ها و bias تنظیم میشوند تا مدل به خروجی بهینه و مورد نظر برسد.

در فرمول زیر محاسبات انجام گرفته را مشاهده میکنید که خروجی این فرمول به عنوان ورودی به تابع فعالسازی داده میشود. در این فرمول یک مجموع وزن دار روی ورودی های نرون صورت میگیرد. این مجموع در نهایت با مقدار bias جمع میشود.

$$\sum_{j=1}^n x_j w_j$$

در ادامه به بررسی توابع فعالسازی میپردازیم. استفاده از این توابع غیرخطی برای بازنمایی های پیچیده و الگو های پیچیده داده ها لازم است چراکه بدون استفاده از این تابع ها تمامی نرون ها یک تبدیل خطی روی داده ها اعمال میکنند و کل شبکه به یک ترکیب خطی از ورودی ها تبدیل میشود و عملا تنها الگوهای ساده و خطی قابل استخراج میشوند. این توابع مانند نرون های مغز فعال میشوند و اتصال خود را با نرون بعدی برقرار میکنند. از توابع فعالسازی متداول که در شبکه های عصبی مورد استفاده قرار میگیرند میتوان به ReLU، leaky ReLU، tanh، sigmoid، softmax اشاره کرد که در ادامه به معرفی هر یک از آنها میپردازیم.

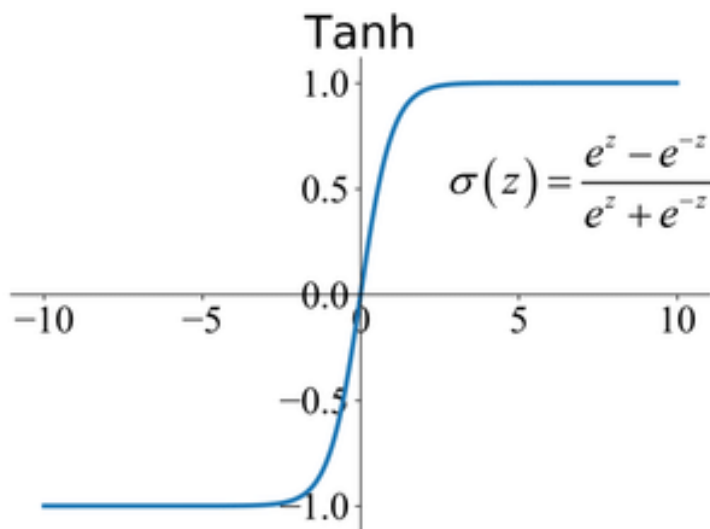
- **تابع فعالسازی ReLU** – تابع Rectified Linear Unit activation function یا به اختصار ReLU تابعی است با نمودار و فرمول زیر:



تصویر شماره ۴

که خروجی آن برای ورودی های بزرگتر از ۰ برابر با همان ورودی است. این تابع متداول ترین تابع در شبکه های عصبی میباشد چراکه فرایند آموزش با این تابع سریعتر هست. این تابع از نظر محاسباتی نیز بسیار کارآمد است و سرعت همگرایی بالایی دارد. این تابع مشکل مرگ نرون را دارد که در مقادیر ورودی نزدیک صفر گرادیان تابع صفر میشود و در آموزش تاثیر میگذارد.

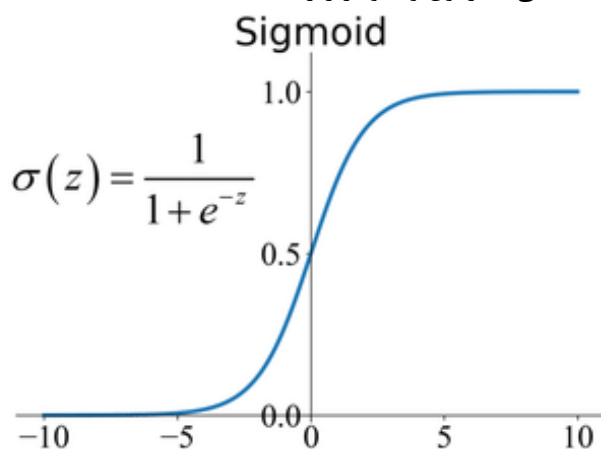
- **تابع فعالسازی tanh** – تابع تانژانت هایپربولیک تابعی است با فرمول و نمودار زیر:



تصویر شماره ۵

که در آن خروجی بین ۱ و -1 بسته به مقدار ورودی تغییر میکند. این تابع مشکل محوشدگی گرادیان را دارد به این معنی که در مقادیر بسیار بزرگ یا کوچک ورودی، مشتق بسیار کوچک میشود و به آموزش شبکه لطمه میزند. این تابع به همین دلیل به کندی همگرا میشود و فرایند آموزش شبکه را طولانی میکند.

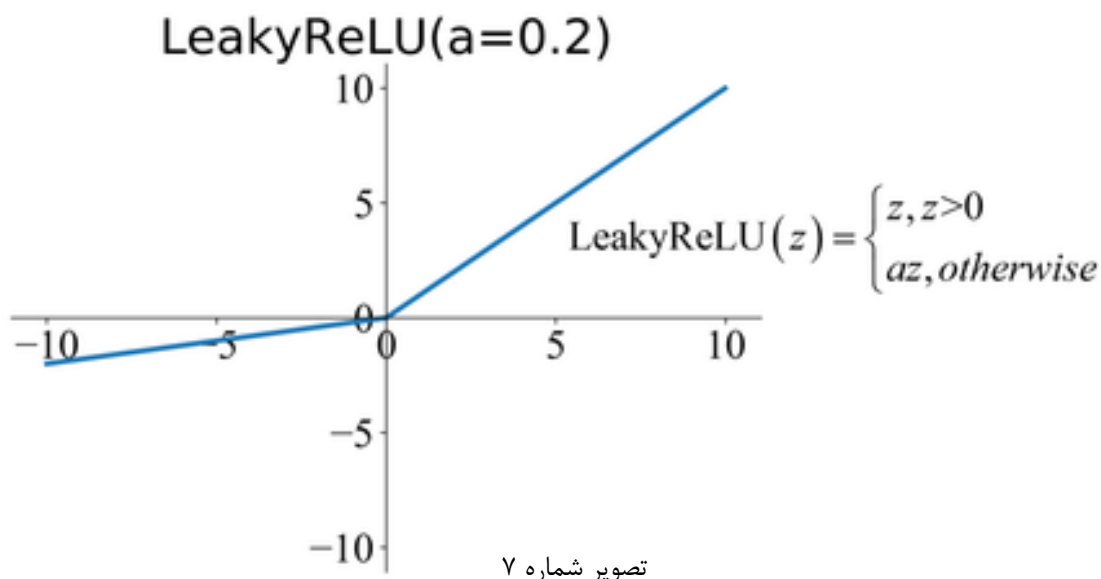
• تابع فعالسازی Sigmoid – تابعی با فرمول و نمودار زیر:



تصویر شماره ۶

این تابع یک منحنی S شکل است. زمانی که می‌خواهیم خروجی مدل احتمال باشد، از تابع سیگموئید استفاده می‌کنیم. چون تابع سیگموئید مقادیر را به بازه صفر تا ۱ می‌برد و احتمالات هم میان همین بازه قرار دارند. این تابع هم مشکل محوشدگی گرادیان و هم گرایبی کند را دارد.

• تابع leaky ReLU – تابعی با فرمول و نمودار زیر:



انتخاب تابع فعالسازی برای نرون ها به نوع مسئله مرتبط میباشد و هر کدام را میتوان با توجه به شرایط استفاده کرد. نکات زیر معمولا را میتوان در رابطه با این توابع فعالسازی بیان کرد:

- تابع سیگموئید در مسائل کلاس بندی معمولا به خوبی عمل میکند.
- توابع سیگموئید و \tanh به دلیل مشکل محوشدگی گرادیان، در برخی موارد استفاده نمیشوند.
- در بیشتر مواقع از تابع ReLU استفاده میشود که این تابع نتایج خوبی را ارائه میکند.
- تابع ReLU تنها در لایه های پنهان مورد استفاده قرار میگیرد چرا که مقدار آن تا بینهایت میرود و ما در خروجی معمولا به یک احتمال یا یک مقدار کراندار نیاز داریم.
- در مواجهه با مشکل مرگ نرون میتوان از تابع leaky ReLU استفاده کرد.

فرایند آموزش شبکه عصبی

فرایند آموزش یا training process به تنظیم کردن وزن های هر نرون برای مینیمم کردن خطای مدل گفته میشود. در این فرایند با استفاده از یک تابع خطا، خطای مدل در پیشبینی خروجی با توجه به مجموعه ورودی های مدل (مجموعه داده های آموزش یا training data) محاسبه میشود و با توجه به این خطا وزن ها تنظیم میشوند. یکی از روش هایی که برای اینکار وجود دارد روش پس انتشار خطا یا back propagation of error نام دارد. در این روش دو مرحله وجود دارد که در مرحله ی اول که حرکت رو به جلو یا feed forward نام دارد، داده های ورودی از لایه ی ورودی به شبکه داده میشوند و این داده ها به سمت لایه خروجی رفته و عملیات هر نرون روی آنها اعمال میشود (مجموع وزن دار و توابع فعالسازی). سرانجام پس از این مرحله یک خروجی بدست میاید که این خروجی مقداری خطا نسبت به خروجی واقعی دارد. اینجا مقدار خطا توسط تابع خطا محاسبه میشود و وارد مرحله دوم میشود. در این مرحله که انتشار رو به عقب یا back propagation نام دارد، وزن ها به هنگام سازی میشوند. این به هنگام سازی با توجه به مقدار تابع خطا صورت میگیرد تا در تکرار های بعدی این عملیات خطای کمتری داشته باشیم. در نهایت

هدف رسیدن به کمترین مقدار خطا میباشد. در مرحله دوم این روش از مقدار خطا شروع به بازگشت به عقب میکنیم تا به لایه ورودی برسیم. در این مسیر مقدار گرادیان ها را محاسبه میکنیم و با داشتن مقدار خطا، سعی در مینیمم کردن مقدار خطا میکنیم. برای اینکار میتوان از روش کاهش گرادیان یا Gradient Descent استفاده کرد که در این روش با محاسبه ی گرادیان تابع خطا وزن ها را بهینه سازی میکند تا مقدار تابع خطا به مقدار مینیمم آن نزدیک شود.

توابع خطا یا توابع زیان (loss function) مختلفی برای استفاده در شبکه های عصبی وجود دارند که میتوان چند نمونه از آن ها را در ادامه مشاهده کرد (در فرمول هایی که در ادامه آورده شده است، \hat{y} نشان دهنده ی مقدار پیشبینی شده توسط شبکه و y نشان دهنده ی مقدار واقعی خروجی میباشد)

- **تابع Means Square Error –** یکی از متداول ترین توابع مورد استفاده در شبکه های عصبی که برای مسائل رگرسیون هستند. این تابع میانگین مربعات خطا نسبت به مقدار واقعی را محاسبه میکند. فرمول این تابع به صورت زیر میباشد:

$$MSE = \frac{\sum (y_i - \hat{y}_i)^2}{n}$$

- **تابع Mean Absolute Error –** از این تابع نیز در مسائل رگرسیون استفاده میشود. این تابع میانگین قدرمطلق تفاضل بین مقدار پیشبینی شده و مقدار واقعی را محاسبه میکند. فرمول این تابع به صورت زیر میباشد:

$$MAE = \frac{\sum |y_i - \hat{y}_i|}{n}$$

- **تابع Binary Cross-entropy –** از این تابع در مسائل کلاس بندی با دو کلاس استفاده میشود. فرمول این تابع به صورت زیر میباشد:

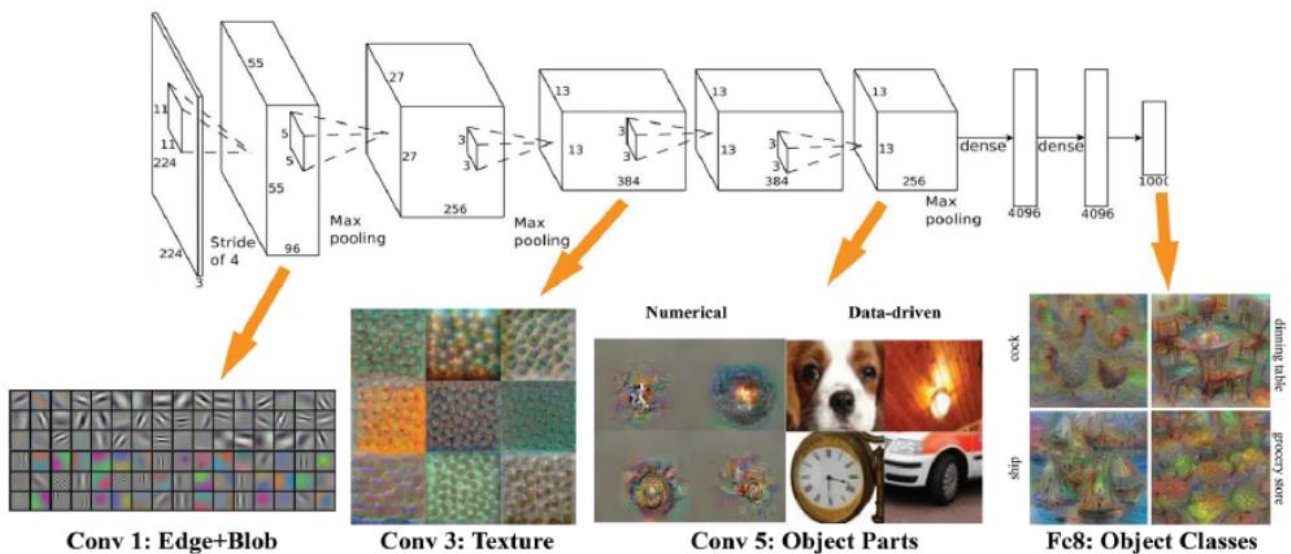
$$-\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

در نهایت با استفاده از این توابع مقدار کلی خطا برای تمامی داده های آموزش با میانگین گیری روی تمام خطا ها محاسبه میشود.
در نهایت با استفاده از روش گرادیان کاهشی و تکرار بر روی داده های آموزش، مقادیر وزن ها برای مینیمم سازی میزان خطا بهینه میشوند تا مدل با خطای کمی خروجی را پیشبینی کند.

۲-۲-۳ شبکه های عصبی کانولوشنال

یکی از مدل های رایج شبکه های عصبی عمیق، شبکه عصبی کانولوشنال یا Convolutional Neural Network یا به اختصار CNN میباشد. این نوع شبکه عصبی برای پردازش تصویر بسیار مناسب میباشد و این شبکه ها یک بازنمایی پیچیده از داده های تصویری را با استفاده از مقدار حجمی از داده ها یاد میگیرند. این شبکه ها از سیستم بینایی انسان الهام گرفته شده اند و چندین لایه از تبدیلات را یاد میگیرند. این شبکه ها از دو نوع لایه تشکیل میشوند که به شرح زیر میباشد:

- **لایه کانولوشن** - این لایه با توجه به اینکه در تصاویر پیکسل ها در کنار هم معنا دار هستند و تک به تک معنایی ندارند و یک locality بین آنها برقرار است، با عملیات کانولوشن بر روی آنها، ویژگی ها را استخراج میکند.
- **لایه تلفیق** - در این لایه ماکزیمم یا متوسط ویژگی ها بر روی یک ناحیه خاص محاسبه میشوند تا اندازه ی تصویر کاهش یابد و محاسبات در لایه های بعدی سبک تر شود.

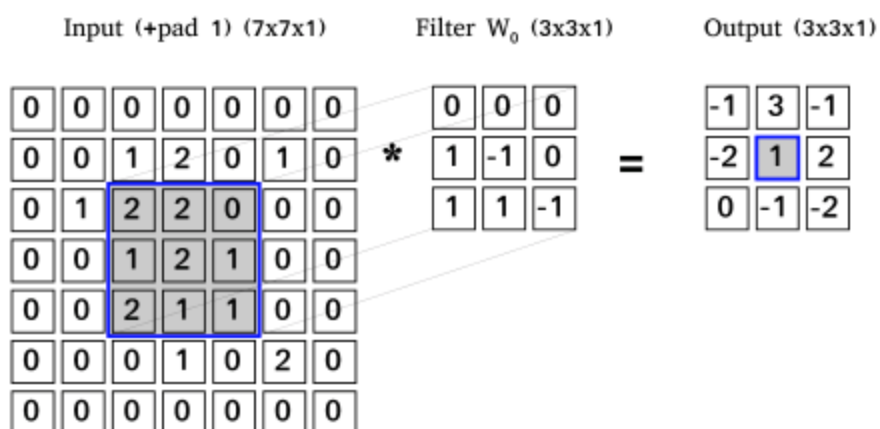


تصویر شماره ۸

هر لایه ی کانولوشن از تعدادی فیلتر تشکیل شده است که هر یک از این فیلتر ها به صورت یک ماتریس دوبعدی از اعداد هستند که در اصل معادل با وزن ها در شبکه عصبی مصنوعی عمل میکنند و در نهایت هدف تنظیم این مقادیر برای بهینه سازی عملکرد شبکه میباشد.

عملیات کانولوشن:

در این عملیات یک ماتریس دوبعدی به عنوان فیلتر بر روی مقادیر پیکسل های تصویر (که خود نیز یک ماتریس دوبعدی میباشد) اعمال میشود. این عملیات سبب میشود تا به جای یادگیری یک وزن به ازای هر پیکسل از تصویر، با یادگیری وزن های فیلتر به عملکردی بهتر و بهینه تر از نظر محاسباتی رسید. در تصویر زیر میتوان نحوه ی عملکرد این عملیات را مشاهده کرد:



تصویر شماره ۹

در تصویر بالا مربع بزرگ مقادیر تصویر و مربع کوچک وسطی مقادیر فیلتر را نشان میدهند. در هر مرحله فیلتر بر روی بخشی از تصویر اعمال میشود و خروجی آن بخش که یک عدد میباشد را در ماتریس خروجی قرار میدهد. این فیلتر با یک گام مشخص (برای مثال دو پیکسل دو پیکسل) روی تصویر حرکت میکند تا تمامی بخش های تصویر را پوشش دهد. نحوه ی محاسبات به این صورت میباشد که هر درایه از ماتریس در درایه متناظر با آن در فیلتر ضرب میشود و در نهایت نتیجه ی تمامی این ها با هم جمع شده و به عنوان خروجی در نظر گرفته میشود. برای مثال برای تصویر بالا داریم:

$$2 \times 0 + 2 \times 0 + 0 \times 0 + 1 \times 1 + 2 \times -1 + 1 \times 0 + 2 \times 1 + 1 \times 1 + 1 \times -1 = 1$$

پس از این که خروجی عملیات کانولوشن در لایه کانولوشن مشخص شد این ماتریس با ماتریس bias جمع شده و نتیجه آن به یک تابع فعالسازی داده میشود و خروجی این تابع فعالسازی به عنوان ورودی لایه بعدی در نظر گرفته میشود.

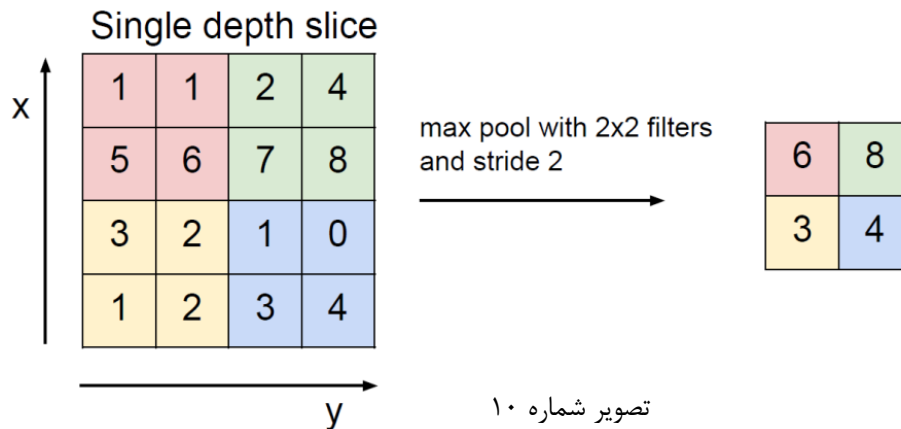
عملیات تلفیق:

از این عملیات برای کاهش ابعاد بازنمایی ها استفاده میشود. روش های مختلفی برای این عملیات وجود دارد که در ادامه به آنها میپردازیم.

- **Max-Pooling** – در این روش هر فیلتر با یک گام مشخص بر روی تصویر حرکت میکند و هر بار مقدار ماکزیمم را به عنوان خروجی باز میگرداند. فرمول آن به صورت زیر میباشد:

$$h_i^n(r, c) = \max_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c})$$

در تصویر زیر میتوان اعمال این عملیات بر یک ماتریس را مشاهده کرد:

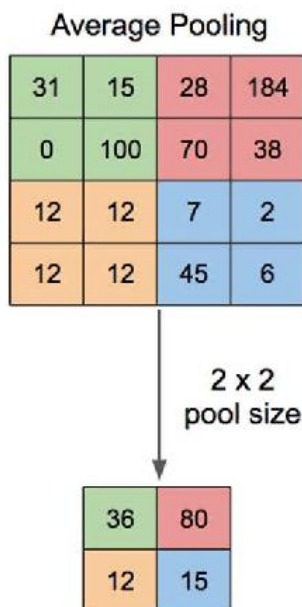


در تصویر بالا یک فیلتر max-pool ۲ در ۲ با اندازه گام ۲ بر روی تصویر اعمال شده که خروجی به صورت ماتریس سمت راست میشود.

- **Average-Pooling** – در این روش هر فیلتر با یک گام مشخص بر روی تصویر حرکت میکند و هر بار مقدار میانگین را به عنوان خروجی باز میگرداند. فرمول آن به صورت زیر میباشد:

$$h_i^n(r, c) = \text{mean}_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c})$$

در تصویر زیر میتوان اعمال این عملیات بر روی یک ماتریس مشاهده کرد:

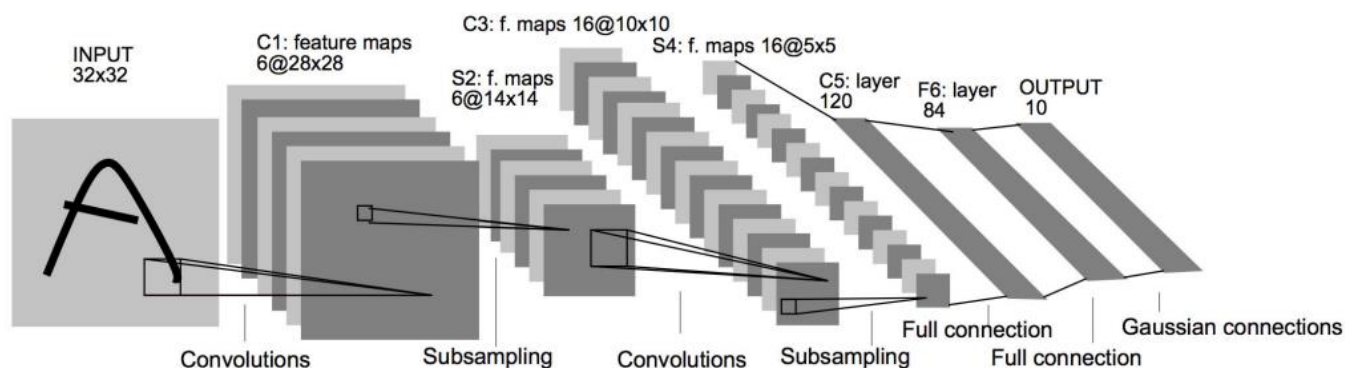


تصویر شماره ۱۱

در تصویر بالا یک فیلتر average-pool ۲ در ۲ با اندازه گام ۲ بر روی تصویر اعمال شده که خروجی به صورت ماتریس پایینی میشود.

این دو روش از متداول ترین روش ها میباشند و معمولاً از یکی از این دو روش در شبکه های کانوولوشنال استفاده میشود.

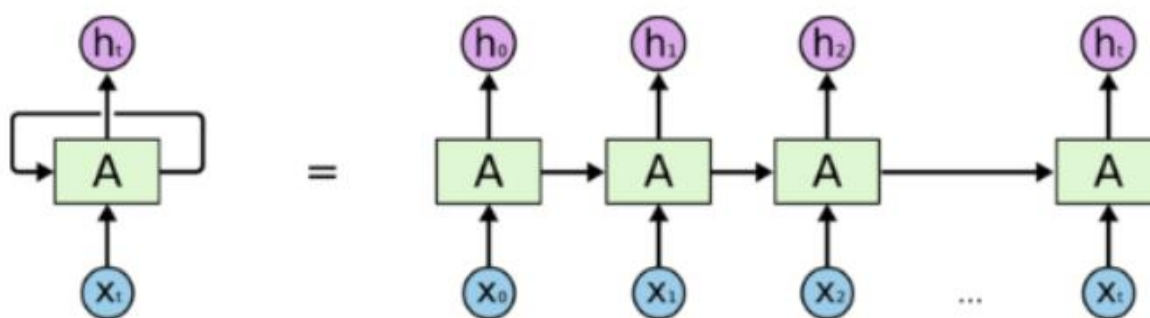
پس از لایه های کانوولوشن و تلفیق، برای بدست آوردن خروجی که به صورت تک عدد میباشد، نمیتوان از فیلتر ها که خروجیشان ماتریسی از اعداد است استفاده کرد و در لایه های آخر شبکه های کانوولوشنال از یک لایه ی تماماً متصل (Fully Connected Layer) استفاده میشود. ورودی این لایه تخت شده ی ماتریس خروجی آخرین لایه ی کانوولوشن یا تلفیقی (بسته به معماری شبکه) میباشد. به این معنا که ماتریس دوبعدی خروجی لایه قبل به یک آرایه تک بعدی تبدیل میشود و این آرایه به عنوان ورودی به این لایه داده میشود. این لایه همانند لایه پنهان در شبکه عصبی مصنوعی میباشد و هر نرون آن یک وزن و یک bias دارد. در شبکه های کانوولوشنال معمولاً چندین لایه ی آخر شبکه لایه های Fully Connected میباشند که در لایه ی خروجی با توجه به نوع مسئله تابع فعالساز مناسب انتخاب میشود. در تصویر زیر یک نمونه از شبکه عصبی کانوولوشنال را مشاهده میکنید:



تصویر شماره ۱۲

۳-۲-۳ شبکه های عصبی بازگشتی

شبکه عصبی بازگشتی Recurrent Neural Network یا به اختصار RNN نوعی از شبکه های عصبی میباشد که در تشخیص گفتار، پردازش زبان طبیعی، پردازش صوت و به صورت کلی پردازش داده های ترتیبی (Sequential data) استفاده میشود. در این شبکه ها برخلاف شبکه های معرفی شده تا اینجا، یک لایه ی بازخورد داریم که خروجی شبکه به همراه ورودی بعدی به شبکه بازگردانده میشود. ای شبکه ها با استفاده از این مکانیزم دارای حافظه داخلی میباشند و میتوانند ورودی های قبلی را به خاطر بسپارند و از این حافظه برای پردازش دنباله ای از ورودی ها بهره ببرند.



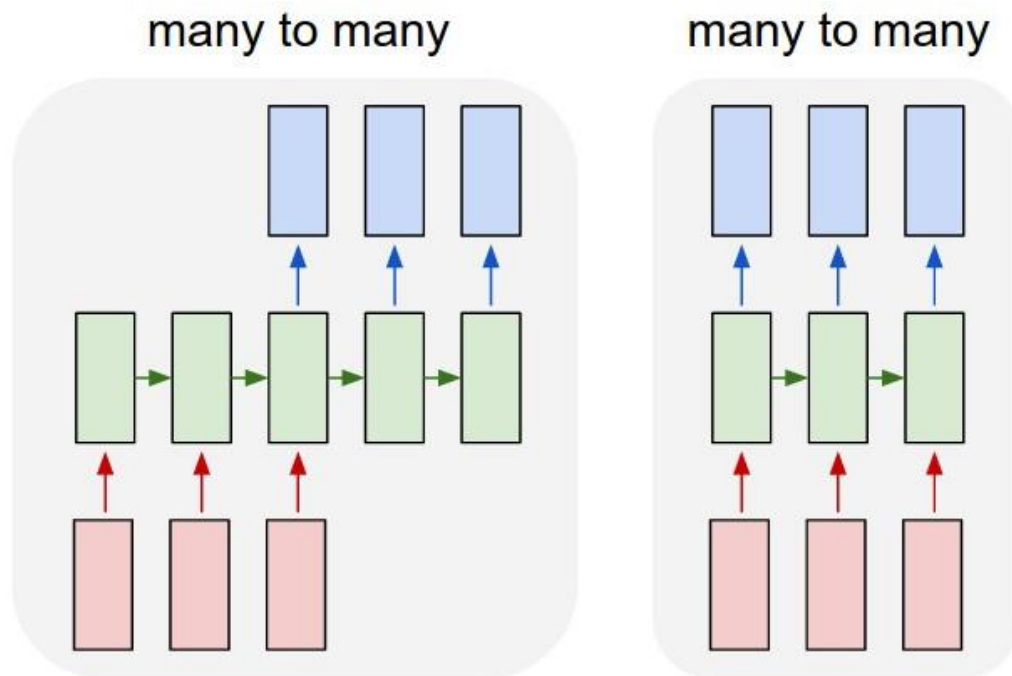
An unrolled recurrent neural network.

تصویر شماره ۱۳

در تصویر بالا یم لایه از شبکه عصبی بازگشتی را مشاهده میکنید که در اصل یک واحد میباشد که از خروجی زمان $t-1$ در زمان t به عنوان ورودی استفاده میکند و با استفاده از مکانیزم هایی میتواند اطلاعاتی از زمان های قبل تر هم نگه دارد.

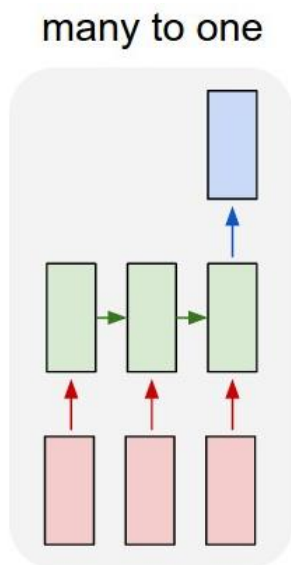
معماری های مختلفی از این شبکه وجود دارد که میتوان به نمونه های زیر اشاره کرد:

- Many-to-many در این معماری هم ورودی و هم خروجی یک دنباله میباشد. از موارد استفاده این معماری میتوان به ترجمه ماشینی اشاره کرد.



تصویر شماره ۱۴

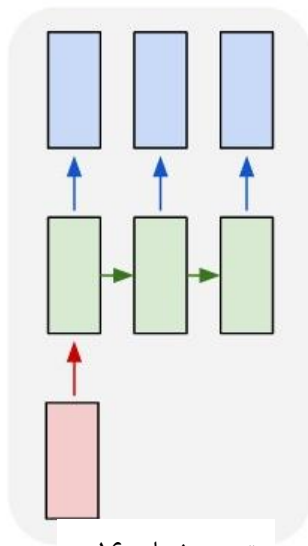
- Many-to-one در این معماری ورودی به صورت یک دنباله و خروجی به صورت یک عدد میباشد. از موارد استفاده ی این معماری میتوان به تحلیل احساسات اشاره کرد که با گرفتن یک جمله که دنباله ای از کلمات است، یک عدد در رابطه با احساس جمله ی ورودی (مثبت یا منفی بودن آن) بازمیگرداند. از دیگر موارد استفاده این معماری که در این پروژه نیز استفاده شده به دسته بندی صوت میتوان اشاره کرد.



تصویر شماره ۱۵

- One-to-many در این معماری ورودی به صورت یک عدد و خروجی یک دنباله میباشد. از موارد استفاده این معماری میتوان به تولید موسیقی با ورودی ژانر آن اشاره کرد.

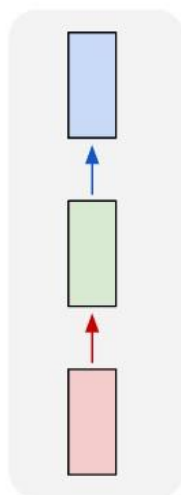
one to many



تصویر شماره ۱۶

- One-to-One در این معماری هم ورودی و هم خروجی یک عدد میباشد. این معماری همانند یک شبکه عصبی مصنوعی عمل میکند.

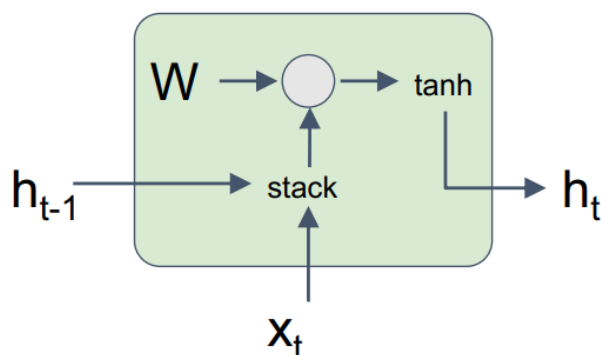
one to one



تصویر شماره ۱۷

در شبکه های عصبی بازگشتی نیز هر نرون حاوی وزن ها و bias و یک تابع فعالسازی میباشد و تنها معماری آن متفاوت میباشد. در این مدل شبکه عصبی معمولا از تابع فعالسازی tanh استفاده میشود.

یک واحد ساده از شبکه عصبی بازگشتی به صورت زیر میباشد:



تصویر شماره ۱۸

فرمول های مورد استفاده در این واحد به صورت زیر میباشد:

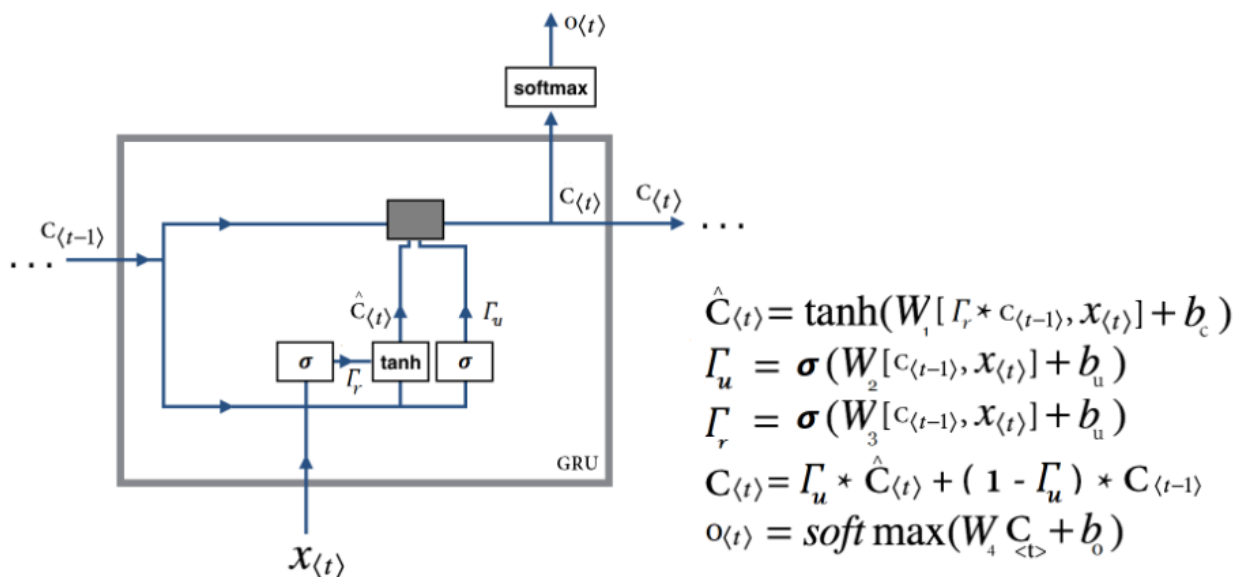
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

که در این فرمول W_{hh} نشان دهنده ی وزن های اعمال شده روی خروجی واحد در زمان قبلی (h_{t-1}) و W_{xh} نشان دهنده وزن های اعمال شده روی ورودی در این زمان (x_t) میباشد. در نهایت پس از اعمال وزن ها از تابع فعالسازی میگذرند و خروجی این واحد را در زمان t تشکیل میدهند.

این مدل ساده تنها از خروجی یک زمان قبل تر استفاده میکند و از مشکل محو شدگی گرادیان رنج میبرند. برای استفاده از اطلاعات زمان های قبل تر باید به این واحد حافظه اضافه کنیم. دو مدل واحد متعارف حافظه دار با نام های LSTM و GRU وجود دارند.

GRU یا Gated Recurrent Unit به استفاده از گیت ها برای نگهداری اطلاعات از گام های زمانی قبلی میباشد. در این واحد گیت های بروزرسانی و بازنشانی (Update gate و Reset gate) که مشخص میکنند چه اطلاعاتی به خروجی منتقل شوند و چه اطلاعاتی منتقل نشوند.

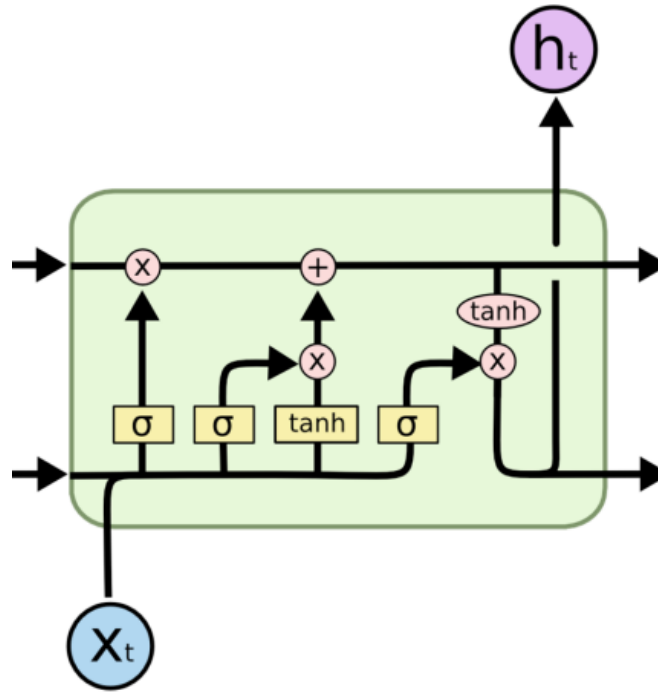
در تصویر زیر میتوان یک واحد GRU و فرمول های مربوط به گیت های آن را مشاهده کرد:



تصویر شماره ۱۹

در اینجا C همان سلول حافظه را نشان میدهد که در اصل همان h_t در واحد ساده میباشد. فرمول مربوط به گیت بروزرسانی با نشان داده میشود و فرمول مربوط به گیت بازنشانی با Γ_r نمایش داده میشود. هر یک Γ_u از این گیت ها وزن های مخصوص به خود را دارند که همگی در فرایند آموزش تنظیم میشوند.

LSTM یا Long Short Term Memory دیگر واحد حافظه دار برای شبکه های بازگشتی میباشد. این واحد هم مانند GRU با استفاده از گیت ها اطلاعات را از زمان های قبلی به خاطر میسپارد. در این واحد یک گیت جدید به نام گیت فراموشی (Forget Gate) وجود دارد که در صورت نیاز به فراموش کردن اطلاعات قبلی منجر میشود. در تصویر زیر میتوان شکل کلی این واحد و فرمول های مربوط به آن را مشاهده کرد:



تصویر شماره ۲۰

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

در فرمول های بالا فرمول اول مربوط به گیت فراموشی و فرمول دوم مربوط به گیت ورودی و فرمول سوم مربوط به حافظه ی داخلی و فرمول چهارم به گیت خروجی و فرمول آخر مربوط به خروجی واحد میباشد. در این پروژه از این واحد در شبکه عصبی بازگشتی

استفاده شده است بدلیل ماهیت مسئله و اینکه برای تشخیص یک کلمه در یک سیگنال صوتی اطلاعات گام های زمانی قبلی در تصمیم گیری تاثیر دارند.

۳-۳ معیارهای سنجش عملکرد مدل های یادگیری ماشین

قبل از بررسی روش ها و معیارها باید به چند مورد که در فرمول های این روش ها استفاده میشود اشاره میکنیم.

- **True Positive:** داده هایی که برای کلاس True هستند (در این پروژه داده هایی که کلمه کلیدی هستند) و مدل به درستی کلاس آنها را تشخیص داده است.
- **False Positive:** داده هایی که برای کلاس False هستند (در این پروژه داده هایی که کلمه کلیدی نیستند) و مدل به اشتباه آنها را بعنوان کلاس True تشخیص داده است.
- **True Negative:** داده هایی که برای کلاس False هستند و مدل به درستی کلاس آنها را تشخیص داده است.
- **False Negative:** داده هایی که برای کلاس True هستند و مدل به اشتباه کلاس آنها را تشخیص داده است.

در ادامه به معیارهای ارزیابی مدل های یادگیری ماشین میپردازیم:

Precision: این معیار دقت مدل برای تشخیص کلاس ۱ را نشان میدهد. این مقدار دقت با تقسیم تعداد نمونه هایی که به درستی تشخیص داده شده اند بر تعداد کل نمونه ها بدست می آید. در تصویر زیر فرمول این معیار را مشاهده میکنید.

$$Precision = \frac{TP}{TP + FP}$$

Recall: این معیار نسبت تعداد نمونه هایی از کلاس ۱ که مدل آنها را به درستی تشخیص داده را به کل نمونه های با کلاس ۱ نشان میدهد.

$$Recall = \frac{TP}{TP + FN}$$

F1-score: این معیار میانگین هارمونیک از دو معیار قبلی را به ما میدهد که با رابطه زیر قابل محاسبه است.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Support: تعداد نمونه ها در هر کلاس را مشخص میکند.

۳-۴ ابزارها

در این بخش به معرفی ابزار های مورد استفاده برای پیاده سازی این پروژه میپردازیم.

برای پیاده سازی این پروژه در تمامی برنامه ها از زبان برنامه نویسی پایتون استفاده شده است. در ادامه کتابخانه های کمکی که استفاده شده اند را معرفی میکنیم.

- **کتابخانه qtpy: PyQt** کتابخانه ای است که به شما امکان می دهد از فریمورک Qt GUI در پایتون استفاده کنید. خود Qt در ++C نوشته شده است که با استفاده از آن از طریق پایتون، می توانید اپلیکیشن ها را با سرعت بسیار بیشتری بسازید. PyQt5 به جدیدترین نسخه پنجم Qt اشاره دارد. ممکن است هنوز هم نام (Py)Qt4 را در وب بیابید، اما این نسخه دیگر قدیمی شده است و پشتیبانی نمی شود.
- چند نخي (multi thread): اولین مفهومی که باید دانست process هستش، process هر برنامه ای هستش که میتونه به شکل مستقل کار کنه مثل Firefox یا Vlc. مفهوم دوم thread هستش، هر thread یک جزء کوچک از process هستش. برای multi threading در پایتون از ماژول threading استفاده میشود. باید توجه داشت که در پایتون نمیتوان multi threading را به معنای واقعی کلمه پیاده سازی کرد. پایتون توانایی اجرای دو thread در یک زمان را ندارد و فقط میتواند برنامه هایی که به شکل I/O bound هستند را مدیریت کند. در صورتی که برنامه به شکل I/O bound باشد زمانی را که صرف منتظر ماندن برای رسیدن پاسخ میکند را میتواند به کار دیگری مشغول شود. اما اگر برنامه CPU bound باشد و همواره cpu مشغول کار باشد استفاده کردن از multi threading نه تنها باعث افزایش سرعت نخواهد شد بلکه باعث افزایش فشار بروی cpu میشود و در نتیجه با کندی سرعت مواجه خواهیم شد.

در فصل های بعدی به توضیح بخش های مختلف پیاده سازی شده و توضیح کد های آنها خواهیم پرداخت.

فصل چهارم

۴-۱ چند نخ

Multithreading توانایی یک برنامه یا یک سیستم عامل برای فعال کردن بیش از یک کاربر در یک زمان بدون نیاز به چندین نسخه از برنامه در حال اجرا در رایانه است. Multithreading همچنین می تواند چندین درخواست از یک کاربر را مدیریت کند. اجرای چندین رشته مشابه اجرای چندین برنامه مختلف به طور همزمان است، اما با مزایای زیر:

- چندین رشته در یک فرآیند، فضای داده مشابهی را با رشته اصلی به اشتراک می گذارند و بنابراین می توانند اطلاعات را به اشتراک بگذارند یا راحت تر از زمانی که فرآیندهای جداگانه باشند، با یکدیگر ارتباط برقرار کنند.
- موضوعاتی که گاهی اوقات فرآیندهای سبک وزن نامیده می شوند و به حافظه زیادی نیاز ندارند. آنها ارزان تر از فرآیندها هستند.

یک thread یک شروع، یک دنباله اجرا و یک نتیجه دارد. این یک نشانگر دستورالعمل دارد که مکان هایی را که در حال حاضر در زمینه اش اجرا می شود، ردیابی می کند.

- می توان آن را از قبل قطع کرد (وقفه)
- می توان آن را به طور موقت در حالت تعلیق قرار داد (همچنین به عنوان خواب شناخته می شود) در حالی که رشته های دیگر در حال اجرا هستند - به این حالت تسلیم می گویند.

۴-۲ شروع یک نخ جدید

برای ایجاد یک رشته دیگر، باید متد زیر موجود در ماژول رشته را فراخوانی می کنیم

```
thread.start_new_thread ( function, args[, kwargs] )
```

این فراخوانی روشی سریع و کارآمد را برای ایجاد موضوعات جدید در لینوکس و ویندوز امکان پذیر می کند.

فراخوانی متد فوراً برمی گردد و رشته فرزند شروع می شود و تابع را با لیست ارسال شده از آرگ ها فراخوانی می کند. هنگامی که تابع برمی گردد، رشته خاتمه می یابد.

در اینجا، `args` مجموعه ای از استدلال است. از یک تاپل خالی برای فراخوانی تابع بدون ارسال هیچ آرگومان استفاده کنید. `kwargs` یک فرهنگ لغت اختیاری از آرگومان های کلیدواژه است.

مثال

```
#!/usr/bin/python

import thread
import time

# Define a function for the thread
def print_time( threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % ( threadName, time.ctime(time.time()) )

# Create two threads as follows
try:
    thread.start_new_thread( print_time, ("Thread-1", 2, ) )
    thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
    print "Error: unable to start thread"

while 1:
    pass
```

هنگامی که کد بالا اجرا می شود، نتیجه زیر را ایجاد می کند

```
Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009
```

اگرچه برای `threading` سطح پایین بسیار موثر است، اما ماژول `thread` در مقایسه با ماژول `threading` جدید بسیار محدود است.

۳-۴ ماژول `Threading`

ماژول `threading` جدیدتر همراه با `Python ۲,۴` پشتیبانی بسیار قدرتمندتر و سطح بالاتری را از `thread` ها نسبت به ماژول `thread` که در بخش قبل مورد بحث قرار گرفت، فراهم می کند.

ماژول `threading` تمام روش های ماژول `thread` را نشان می دهد و برخی روش های اضافی را ارائه می دهد -

`threading.activeCount()` - تعداد اشیاء رشته فعال را برمی گرداند.

`threading.currentThread()` - تعداد اشیاء رشته را در کنترل رشته تماس گیرنده برمی گرداند.

`threading.enumerate()` - لیستی از تمام اشیای رشته را که در حال حاضر فعال هستند را برمی گرداند.

علاوه بر متدها، ماژول `threading` دارای کلاس `Thread` است که `threading` را پیاده سازی می کند. متدهای ارائه شده توسط کلاس `Thread` به شرح زیر است :

run() : متد run() نقطه ورودی یک رشته است.

start() : متد start() یک رشته را با فراخوانی متد run شروع می کند.

join():join([time]) () منتظر می ماند تا رشته ها خاتمه پیدا کنند.

isAlive() : متد isAlive() بررسی می کند که آیا یک رشته هنوز در حال اجرا است یا خیر.

getName() : متد getName() نام یک رشته را برمی گرداند.

setName() : متد setName() نام یک رشته را تنظیم می کند.

۴-۴ توضیح کد app.py

در ادامه به توضیح کد app.py میپردازیم.

```
import pyaudio
import numpy as np
from librosa import load
from librosa.feature import mfcc as mf
import time
import soundfile as sf
import tensorflow.keras as keras
import matplotlib.pyplot as plt
from threading import Thread
import os
```

تصویر شماره ۲۱

ابتدا کتابخانه های مورد استفاده در این کد را به آن اضافه میکنیم.

- کتابخانه pyaudio که از آن برای دریافت ورودی از میکروفون استفاده شده است.
- تابع load از کتابخانه librosa که برای خواندن سیگنال های صوتی از فایل ها استفاده شده است.
- تابع mfcc از کتابخانه librosa که برای استخراج ضرایب mfcc از سیگنال صوتی استفاده میشود.
- کتابخانه soundfile که برای ذخیره فایل های صوتی روی سیستم استفاده شده است.
- کتابخانه keras که برای استفاده از مدل ها به آن نیاز داریم.
- کتابخانه matplotlib که برای ترسیم نمودار ها از آن استفاده میشود.
- کتابخانه threading که برای چند نخه مورد استفاده قرار گرفته است.

```
class Recorder:
    def __init__(self, sample_rate=22050, record_seconds=1):
        self.model = keras.models.load_model('rnnmodel.h5')
        self.queue = list()
        self.chunk = 1050
        self.sample_rate = sample_rate
        self.record_seconds = record_seconds
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=pyaudio.paFloat32,
                                   channels=1,
                                   rate=self.sample_rate,
                                   input=True,
                                   output=True,
                                   frames_per_buffer=self.chunk)
```

در این قسمت کلاس Recorder تعریف شده است که تمامی کار های برنامه که مربوط به بخش یادگیری ماشین میباشد با این کلاس است. در تصویر بالا تابع `__init__` از این کلاس را مشاهده میکنید که این تابع همان تابع `constructor` در زبان پایتون میباشد و هر زمان ما یک نمونه جدید از این کلاس ایجاد میکنیم این تابع اجرا میشود. در این تابع مقداردهی اولیه خصوصیات نمونه را انجام میدهیم. ورودی های این تابع نرخ نمونه برداری و طول فایل های صوتی مورد استفاده در این برنامه میباشد. پس از آن دیگر مقادیر مقداردهی میشوند:

- `model` که مدل مورد استفاده توسط برنامه را نشان میدهد. نوع این مدل تا زمانی که ورودی و خروجی آن بصورت استاندارد باشد (ماتریسی با ابعاد ۴۳ در ۱۳) تفاوتی ندارد و برای این مسئله میتوان هم از شبکه `cnn` و شبکه `rnn` استفاده کرد. این مدل با استفاده از تابع `load_model` میتواند از فایل ذخیره شده آن، خوانده شود.
- `queue` که لیستی میباشد که سیگنال صوتی ورودی که از میکروفون گرفته میشود در انتهای آن اضافه میشود.
- `Chunk` که تعداد فریم های خوانده شده در هر بار گرفتن ورودی از میکروفون را مشخص میکند.
- `Sample_rate` که نرخ نمونه برداری مورد استفاده را مشخص میکند.
- `Record_seconds` که طول هر فایل صوتی مورد استفاده توسط مدل (به ثانیه) را مشخص میکند.
- `P` و `stream` که با استفاده از کتابخانه `pyaudio` برای گرفتن ورودی از میکروفون استفاده میشوند.

```
def listen(self):
    global record
    global graph
    while record:
        #print('started recording')
        # change this to button press
        ...

        if kb.is_pressed('s'):
            break
        ...

        data = self.stream.read(self.chunk , exception_on_overflow=False)
        self.queue.append(np.frombuffer(data,dtype=np.float32))
        time.sleep(0.01)
    return self.queue
```

تابع `listen` از این کلاس که برای ضبط صوت استفاده میشود. در ابتدا از دو متغیر `global` استفاده شده که برای کنترل حالات برنامه از آنها استفاده میشود. برای مثال متغیر `record` توسط گزینه های `start recording` و `end recording` در بخش گرافیکی برنامه کنترل میشود تا زمان شروع و پایان گرفتن ورودی از میکروفون را توسط آن مشخص کنیم. قبل از پیاده سازی و برای تست این قابلیت همانطور که در قسمت کامنت شده این تابع مشاهده میکنید، از فشردن کلید `s` از کیبورد برای شروع و توقف ضبط صدا استفاده میشود. اما با پیاده سازی بخش گرافیکی برنامه، با استفاده از متغیر `record` این موضوع کنترل میشود. برای ضبط صدا از طریق میکروفون در یک حلقه، سیگنال صوتی توسط تابع `read` از استریم ایجاد شده در قسمت قبلی خوانده میشود و به لیست `queue` اضافه میشود. این کار تا جایی که مقدار متغیر `record` برابر با `True` هست ادامه میابد.

در انتها پس از اینکه مقدار متغیر `record` برابر با `False` شد، صف سیگنال های خوانده شده بازگردانده میشود.

```
def predict(self, path=''):

    if path != '':
        widgets['record_status'][-1].setText('Status: Processing')
        data, _ = load(path)
    else:
        widgets['record_status'][-1].setText('Status: Listening')
        data = self.listen()
        data = np.array(data).flatten()
    ds = []
    # chop up the recorded audio into 1 second signals by hop size = 0.5 seconds
    for i in range(0, len(data), 512):
        if i + 22050 <= len(data):
            ds.append (data[i:i+22050])
    preds = []
    for i in ds:
        mfcc = mf(y=i, sr=22050, n_mfcc=13, n_fft=2048, hop_length=512).T
        mfcc = mfcc[np.newaxis, ..., np.newaxis ]
        preds.append(np.argmax(self.model.predict(mfcc)))

    self.visualize(data, preds)
```

تابع `predict` که کار اصلی این کلاس را انجام میدهد. این تابع برای پیشبینی روی سیگنال های صوتی مورد استفاده قرار میگیرد. ورودی این تابع متغیر `path` میباشد که مقدار پیش فرض آن تهی میباشد. اگر این متغیر مقدار داشت، به این معنی میباشد که سیگنال باید از یک فایل روی سیستم خوانده شود و نه از میکروفون (این برنامه دو قابلیت دارد که یکی تشخیص کلمه کلیدی یک فایل صوتی روی سیستم و دیگری ضبط صوت و تشخیص کلمه کلیدی در آن میباشد). اگر این متغیر مقدار داشت، سیگنال صوتی که به عنوان ورودی به مدل داده میشود توسط تابع `load` از کتابخانه `librosa` خوانده میشود و در متغیر `data` ذخیره میشود و وضعیت برنامه در `processing` قرار داده میشود. اگر متغیر ورودی تابع مقدار نداشت، وضعیت برنامه به `Listening` تغییر داده میشود و تابع `listen` از همین کلاس صدا زده میشود تا صوت ورودی ضبط شود. پس از بازگردانده شدن سیگنال صوتی توسط این تابع، مقادیر موجود در لیست بازگشتی به آرایه نامپای تبدیل میشود و در متغیر `data` ذخیره میشود. پس از اینکه ورودی مدل مشخص شد، ابتدا این سیگنال به سیگنال هایی به طول ۱ ثانیه با گام هایی به اندازه ۰,۵ ثانیه تقسیم میشوند. سپس یک لیست به نام `preds` ایجاد شده که در این لیست کلاس پیشبینی شده برای هر قسمت از این سیگنال ذخیره میشود. در یک حلقه بر روی قسمت های یک ثانیه ای بدست آمده از سیگنال، ضرایب `mfcc` از سیگنال ها با استفاده از تابع `mfcc` از کتابخانه `librosa` استخراج شده و توسط تابع `predict` مدل، کلاس آنها تشخیص داده میشود.

در نهایت تابع `visualize` از همین کلاس صدا زده میشود تا خروجی برای کاربر نمایش داده شود.

```
def visualize(self,data,predictions):
    fig,ax = plt.subplots(figsize=(10,5))
    ax.set_title('Audio Signal')
    ax.set_xlabel('time')
    ax.set_ylabel('amplitude')
    ax.set_facecolor((0.01,0,0.02))
    ax.plot(np.arange(len(data)),data,color='white',lw=0.7)
    for i in range(len(predictions)):
        if predictions[i] == 1:
            y = data[i*512:i*512+22050]
            sf.write(os.getcwd()+'\\'+str(i)+'.wav',y,22050)
            x = np.arange(i*512,i*512+22050)
            if i == 0:
                ax.plot(x,y,color='green',lw=0.7)
            else:
                ax.plot(x,y,color='green',lw=0.7)

    fig.savefig(os.getcwd()+'\\1.png',dpi=500/fig.get_size_inches()[1])
```

تابع `visualize` برای تصویرسازی خروجی و ذخیره تصویر نمودار خروجی که برای کاربر نمایش داده میشود، پیاده سازی شده است. ورودی این تابع سیگنال صوتی اصلی و لیست پیشبینی های انجام شده توسط مدل میباشد. ابتدا یک `figure` از کتابخانه `matplotlib` ایجاد میشود تا نمودار روی آن رسم شود. سپس نام نمودار و نام هر محور مشخص شده و رنگ پس زمینه نمودار نیز مشخص میشود. برای رسم نمودار سیگنال صوتی، ابتدا سیگنال اصلی با رنگ سفید رسم شده و برای هر بخش از سیگنال که کلمه کلیدی در آن تشخیص داده شده، با رنگ سبز روی نمودار قبلی رسم میشود. برای اینکار روی لیست `predictions` یک حلقه ایجاد میکنیم و اگر مقدار هر عضو از این لیست ۱ بود (کلمه کلیدی در آن تشخیص داده شده بود) نمودار سبز را رسم میکنیم. موقعیت این نمودار روی نمودار اصلی نیز توسط این موضوع که هر قسمت طولی به اندازه ۱ ثانیه (۲۲۰۵۰ نمونه) و فاصله هر قسمت با قسمت های بعد و قبل آن ۰.۵ ثانیه (۱۱۰۲۵ نمونه) میباشد، مشخص میشود. این قسمت ها از سیگنال اصلی نیز با استفاده از تابع `write` از کتابخانه `soundfile` در همان فولدر از کد بعنوان فایل صوتی با فرمت `wav`. ذخیره میشود. در نهایت این نمودار در فولدر کد ذخیره میشود. دلیل اینکار هم برای تبادل اطلاعات (تصویر این نمودار) بین نخ ها میباشد. از آنجا که نمودار های کتابخانه `matplotlib` قابل انتقال بین نخ ها نیستند و اگر روی نخ اصلی ایجاد نشده باشند، امکان نمایش آنها وجود ندارد. برای حل این مشکل ابتدا این نمودار را بعنوان یک فایل روی سیستم ذخیره میکنیم و در زمان نمایش آن در رابط گرافیکی که روی نخ اصلی اجرا میشود، آن فایل را میخوانیم. در پیوست میتوانید تصاویری از این خروجی ها را مشاهده کنید.

```
# ----- GUI -----
import sys
from PyQt5.QtWidgets import QApplication, QLabel,
    QPushButton, QVBoxLayout, QWidget, QFileDialog, QGridLayout, QFileDialog
from PyQt5.QtGui import QPixmap
from PyQt5 import QtGui, QtCore
from PyQt5.QtGui import QCursor
```

در این قسمت از برنامه، رابط کاربری گرافیکی پیاده سازی شده است. ابتدا توابع و کتابخانه های مورد استفاده در این قسمت را به کد اضافه میکنیم. برای پیاده سازی رابط گرافیکی از کتابخانه PyQt5 استفاده شده که widget های مورد استفاده در این کد به آن اضافه شده است.

```
app = QApplication(sys.argv)
window = QWidget()
window.setWindowTitle('Keyword Detector')
window.setFixedWidth(1100)
window.setFixedHeight(800)
window.setStyleSheet(
    "background: #161219;"
)

grid = QGridLayout()
```

ابتدا یک نمونه اپلیکیشن گرافیکی توسط تابع QApplication از کتابخانه PyQt5 ایجاد میشود. نام برنامه، طول و عرض صفحه باز شده مقدار دهی شده است. در انتها یک grid با استفاده از تابع QGridLayout ایجاد میشود که چینش المنت های برنامه را مشخص میکند.

```

# ----- recording page -----
def recording_frame():

    #page heading
    heading = QLabel('Recorder')
    heading.setAlignment(QtCore.Qt.AlignCenter)
    heading.setStyleSheet(
        '''
            font-size:30px;
            color:white;
            border-bottom: 1px solid white;
            padding-bottom:10px;
            font-weight:bold;
        '''
    )
    widgets['recorder_heading'].append(heading)

    # page description
    desc = QLabel('Click on "start recording" to record you
    desc.setAlignment(QtCore.Qt.AlignCenter)
    desc.setWordWrap(True)
    desc.setStyleSheet('''
        font-size:18px;
        color:white;
    ''')
    widgets['record_description'].append(desc)

```

تابع `recording_frame` که با اجرای آن محتویات و رابط گرافیکی را روی صفحه ایجاد شده می آورد. ابتدا یک متن بعنوان تیتر این صفحه به آن اضافه میکنیم. اینکار با استفاده از تابع `QLabel` انجام میگردد. سپس با استفاده از تابع `setAlignment` این المنت را به وسط صفحه می آوریم. در کتابخانه `PyQt5` استایل هر المنت با سینتکسی همانند `CSS` مشخص میشود. برای مشخص کردن استایل میتوان از تابع `setStyleSheet` استفاده کرد. ورودی این تابع استایل مورد نظر ما برای این المنت با سینتکس `CSS` میباشد.

در ادامه متن توضیحات استفاده از برنامه اضافه میشود. اینکار نیز با اضافه کردن یک QLabel انجام میگردد.

```
#start recording button
startbutton = QPushButton('Start Recording')
startbutton.setStyleSheet('''
    *{
        margin:10px;
        border-radius:10px;
        padding:10px;
        color:white;
        border:2px solid green;
        font-size: 16px;
        max-width:150px;
    }
    *:hover{
        background:green;
    }
''')
startbutton.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
startbutton.clicked.connect(start_recording)
widgets['record_start_button'].append(startbutton)
```

در قسمت بعدی دکمه ی Start Recording که از آن برای شروع ضبط صدا استفاده میشود تعریف میشود. این المنت با استفاده از تابع QPushButton ایجاد میشود. برای اضافه کردن یک تابع callback (که در زمانی که بر روی این دکمه کلیک شد صدا زده شود) از تابع connect استفاده شده است.


```

#stop recording button
stopbutton = QPushButton('Stop Recording')
stopbutton.setStyleSheet('''
    *{
        margin:10px;
        border-radius:10px;
        padding:10px;
        color:white;
        font-size: 16px;
        max-width:150px;
        border: 2px solid red;
    }
    *:hover{
        background:red;
    }
''')
stopbutton.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
stopbutton.clicked.connect(end_loop)
widgets['record_stop_button'].append(stopbutton)

```

در ادامه نیز دکمه Stop Recording برای توقف ضبط صدا مانند قسمت قبل ایجاد میشود.

```

#select file button
button = QPushButton('Select File')
button.setStyleSheet('''
    *{
        margin:10px;
        border-radius:10px;
        padding:10px;
        color:white;
        font-size: 16px;
        max-width:150px;
        border: 2px solid white;
    }
    *:hover{
        background:white;
        color:black
    }
''')
button.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
button.clicked.connect(dialog)

```

در این بخش از کد دکمه ی Select File ایجاد میشود که برای انتخاب فایل صوتی بعنوان ورودی برنامه استفاده میشود.

```

#process file button
button2 = QPushButton('Process File')
button2.setStyleSheet('''
    *{
        margin:10px;
        border-radius:10px;
        padding:10px;
        color:white;
        font-size: 16px;
        max-width:150px;
        border: 2px solid white;
    }
    *:hover{
        background:white;
        color:black
    }
''')
button2.setCursor(QCursor(QtCore.Qt.PointingHandCursor))
button2.clicked.connect(show_results)

```

در ادامه دکمه ی Process File ایجاد میشود که از آن برای پیشبینی توسط مدل و پردازش فایل صوتی استفاده میشود.

```
# recording status
stat = QLabel('Status: Waiting')
stat.setAlignment(QtCore.Qt.AlignCenter)
stat.setWordWrap(True)
stat.setStyleSheet('''
    font-size:14px;
    color:white;
    font-weight:bold;
    border: 1px solid white;
    padding:5px;
    max-width:100px;
''')
widgets['record_status'].append(stat)
```

در این قسمت یک المنت متنی که وضعیت برنامه را نشان میدهد ایجاد میشود. مقدار متن آن بصورت پیشفرض Waiting میباشد و با انجام عملیات مختلف توسط کاربر، متن آن تغییر میکند.

```
grid.addWidget(widgets['recorder_heading'][-1],0,0,1,3)
grid.addWidget(widgets['record_description'][-1],1,0,1,3)
grid.addWidget(widgets['record_start_button'][-1],2,0)
grid.addWidget(widgets['record_stop_button'][-1],2,1)
grid.addWidget(widgets['record_status'][-1],2,2)
grid.addWidget(button,3,0)
grid.addWidget(button2,3,1)
```

در انتهای تابع تمامی این بخش ها را به grid اضافه میکنیم. برای اینکار از تابع addWidget استفاده شده است که در ورودی اول خود widget و در ورودی دوم و سوم موقعیت آن مشخص میشود. ورودی دوم تعداد سطر های این widget و ورودی سوم تعداد ستون های مورد استفاده توسط هر widget را نشان میدهد.

در ادامه به توابع callback پیاده سازی شده میپردازیم.

```
#----- callback functions for gui -----

def start_recorder(path=''):
    recorder = Recorder()
    recorder.predict(path=path)
```

اولین تابع start_recorder نام دارد که برای ایجاد نمونه جدید از کلاس Recorder و پردازش سیگنال صوتی استفاده میشود.

```
def start_recording(path):
    global record
    if record:
        return
    record = True
    t1 = Thread(target=start_recorder, daemon=True)
    t1.start()
```

تابع start_recording که با فشردن دکمه Start Recording در رابط گرافیکی، یک نخ جدید ایجاد میشود که بر روی این نخ تابع قبلی اجرا میشود.

```

def end_loop(ispath=False):
    global record
    if record == False and ispath == False:
        return
    if record:
        record = False
    widgets['record_status'][-1].setText('Status: Waiting')
    image = 'loading'
    while True:
        if '1.png' in os.listdir():
            break
    time.sleep(0.7)
    image = QPixmap('1.png')
    plot = QLabel()
    plot.setPixmap(image)
    plot.setAlignment(QtCore.Qt.AlignCenter)
    plot.setStyleSheet('margin-top:20px;')
    grid.addWidget(plot,4,0,2,3)
    os.remove(os.getcwd()+ '\\1.png')

```

تابع `end_loop` که با فشردن دکمه Stop Recording ضبط صدا را متوقف میکند. اینکار با قرار دادن مقدار `False` در متغیر `record` صورت میگیرد که پیش تر نحوه استفاده از آن مشخص شد. سپس وضعیت برنامه در `Waiting` قرار میگیرد و فایل تصویر نمودار ذخیره شده توسط تابع `visualize` از روی سیستم خوانده میشود و سپس با ایجاد یک المنت جدید و اضافه کردن این تصویر با استفاده از تابع `QPixmap` به این المنت و اضافه کردن آن به `grid`، خروجی به کاربر نمایش داده میشود.

```

def dialog():
    file , check = QFileDialog.getOpenFileName(None, "QFileDialog.getOpenFileName()",
                                                "", "All Files (*)")

    print(file)
    if check:
        start_recorder(file)

```

تابع `dialog` که با اجرای آن یک پنجره برای انتخاب فایل از روی سیستم باز شده و سپس تابع `start_recorder` با آدرس فایل انتخاب شده صدا زده میشود.

```
recording_frame()

window.setLayout(grid)

window.show()
sys.exit(app.exec())
```

در انتهای کد نیز تابع `recording_frame` را صدا میزنیم تا تمامی ویجت های برنامه به پنجره برنامه اضافه شوند و با تابع `show` به نمایش در بیایند.

۴-۵ توضیح کد `real_time.py`

در ادامه به توضیح کد `real_time.py` میپردازیم. این کد برای آشنایی بیشتر با چند نخ و نحوه استفاده از آن در این مسئله پیاده سازی شده است.

```
import pyaudio
import threading
import time
import wave
import numpy as np
from threading import Event
import tensorflow.keras as keras
import tensorflow as tf
import librosa
import soundfile as sf
```

ابتدا کتابخانه های مورد استفاده در این کد را به آن اضافه میکنیم. کتابخانه `threading` برای چند نخ و کتابخانه `keras` برای استفاده از مدل ها به کد اضافه شده اند.

```
class Listener:

    def __init__(self, sample_rate=22050, record_seconds=2):
        self.chunk = 1050
        self.sample_rate = sample_rate
        self.record_seconds = record_seconds
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=pyaudio.paInt16,
                                   channels=1,
                                   rate=self.sample_rate,
                                   input=True,
                                   output=True,
                                   frames_per_buffer=self.chunk)
```

در این قسمت کلاس Listener پیاده سازی شده که وظیفه آن دریافت ورودی از طریق میکروفون و ذخیره آن میباشد. ابتدا تابع `__init__` برای این کلاس نوشته شده است. در این تابع پارامترهای این کلاس مقداردهی میشوند.

```
def listen(self, queue):
    while True:
        data = self.stream.read(self.chunk , exception_on_overflow=False)
        queue.append(data)
        time.sleep(0.01)
```

تابع `listen` برای این کلاس که ورودی را از استریم ایجاد شده برای این کلاس از میکروفون دریافت میکند و به انتهای صف اضافه میکند. این صف بعنوان ورودی به این تابع داده میشود. برای این کار از تابع `read` که برای استریم های ایجاد شده با کتابخانه `pyaudio` تعریف میشود، استفاده شده است.

```
def run(self, queue):
    thread = threading.Thread(target=self.listen, args=(queue,), daemon=True)
    thread.start()
    print("\nWake Word Engine is now listening... \n")
```

تابع `run` که با ایجاد یک نخ جدید، تابع `listen` از همین کلاس را روی این نخ اجرا میکند.


```
class WakeWordEngine:

    def __init__(self, model_file):
        self.c = 0
        self.model = keras.models.load_model('model13.h5')
        self.listener = Listener(sample_rate=22050, record_seconds=2)
        self.audio_q = list()
```

در ادامه کلاس WakeWordEngine تعریف شده است که از این کلاس برای تشخیص کلمه کلیدی استفاده میشود. ابتدا تابع `__init__` را داریم که پارامترهای نمونه های این کلاس را مقداردهی میکند. ابتدا مدل این کلاس با استفاده از تابع `load_model` از کتابخانه `keras`، از فایل آن خوانده میشود. سپس یک لیست با نام `audio_q` ایجاد میشود که سیگنال های صوتی در آن ذخیره میشود. یک نمونه از کلاس `Listener` نیز برای این کلاس ایجاد میشود تا با استفاده از آن بتواند صدای محیط را از طریق میکروفون دریافت کند.

```
def save(self, waveforms, fname="wakeword_temp"):
    wf = wave.open(fname, "wb")
    # set the channels
    wf.setnchannels(1)
    # set the sample format
    wf.setsampwidth(self.listener.p.get_sample_size(pyaudio.paInt16))
    # set the sample rate
    wf.setframerate(22050)
    # write the frames as bytes
    wf.writeframes(b"".join(waveforms))
    # close the file
    wf.close()
    return fname
```

تابع `save` برای این کلاس که سیگنال های صوتی دریافت شده از میکروفون را در فایل روی سیستم ذخیره میکند تا بعدتر توسط تابع `predict` خوانده شوند و توسط مدل پردازش شوند. بعدتر برای برنامه اصلی از یک لیست مشترک بین نخ ها برای تبادل اطلاعات بین آنها استفاده میکنیم اما در این کد از این روش برای اشتراک اطلاعات بین چند نخ استفاده شده است که با توجه به حجم کمی که این فایل های صوتی دارند، روی سرعت کد تاثیری نمیگذارد. این ذخیره کردن سیگنال بعنوان یک فایل با استفاده از کتابخانه `wave` صورت گرفته است. در انتهای این تابع نام فایل ذخیره شده بازگردانده میشود.

```
def predict(self, audio):
    fname = self.save(audio)
    signal, _ = librosa.load(fname) # don't normalize on train
    signal = signal[:22050]
    mfcc = librosa.feature.mfcc(y=signal, sr=22050, n_mfcc=13, n_fft=2048, hop_length=512).T
    mfcc = mfcc[np.newaxis, ..., np.newaxis ]

    preds = self.model.predict(mfcc)
    idx = np.argmax(preds)
    if idx == 1:
        sf.write('wavs\\'+str(self.c)+'wav', signal, 22050)
        self.c += 1
        print(str(self.c)+': ', end='')
        print(idx)
    return idx
```

تابع predict که سیگنال صوتی خام را بعنوان ورودی دریافت کرده و با استخراج ضرایب mfcc توسط تابع mfcc از کتابخانه librosa آنرا به ورودی قابل استفاده توسط مدلها تبدیل میکند. سپس این ورودی را به مدل میدهد و با استفاده از تابع predict خروجی مدل برای این ورودی را دریافت میکند. اگر مدل کلمه کلیدی را تشخیص داده بود در کنسول ۱ را چاپ کرده و اگر تشخیص نداده بود ۰ چاپ میشود. در انتهای تابع کلاس پیشبینی شده توسط مدل بعنوان خروجی بازگردانده میشود.

```
def inference_loop(self, action):
    while True:
        if len(self.audio_q) > 21: # remove part of stream
            diff = len(self.audio_q) - 21
            for _ in range(diff):
                self.audio_q.pop(0)
            self.predict(self.audio_q)
        elif len(self.audio_q) == 21:
            self.predict(self.audio_q)
        time.sleep(0.05)
```

تابع inference_loop که یک حلقه بینهایت میباشد و با دریافت ورودی از میکروفون، مدام تابع predict را بر روی صوت ورودی اجرا میکند.

۴-۶ مزایای Multithreading در پایتون

مزایای ایجاد یک برنامه چند رشته ای در پایتون به شرح زیر است:

- استفاده موثر از منابع سیستم کامپیوتری را تضمین می کند.
- برنامه های چند رشته ای پاسخگوتر هستند.
- این منابع و وضعیت خود را با موضوعات فرعی (فرزند) به اشتراک می گذارد که آن را اقتصادی تر می کند.
- به دلیل شباهت، معماری چند پردازنده را موثرتر می کند.
- با اجرای همزمان چندین رشته در زمان صرفه جویی می کند.
- سیستم برای ذخیره چندین رشته به حافظه زیادی نیاز ندارد.

فصل پنجم

۵-۱ ارائه معماری برای مدل ها

در این قسمت به جزئیات و معماری مدل ها میپردازیم. برای شروع کار از یک معماری معمول و متداول برای شبکه ها استفاده کردیم که جلوتر به جزئیات آن اشاره میشود. سپس به بررسی عملکرد مدل ها میپردازیم و در نهایت راه حل های استفاده شده برای بهبود عملکرد مدل را بررسی میکنیم.

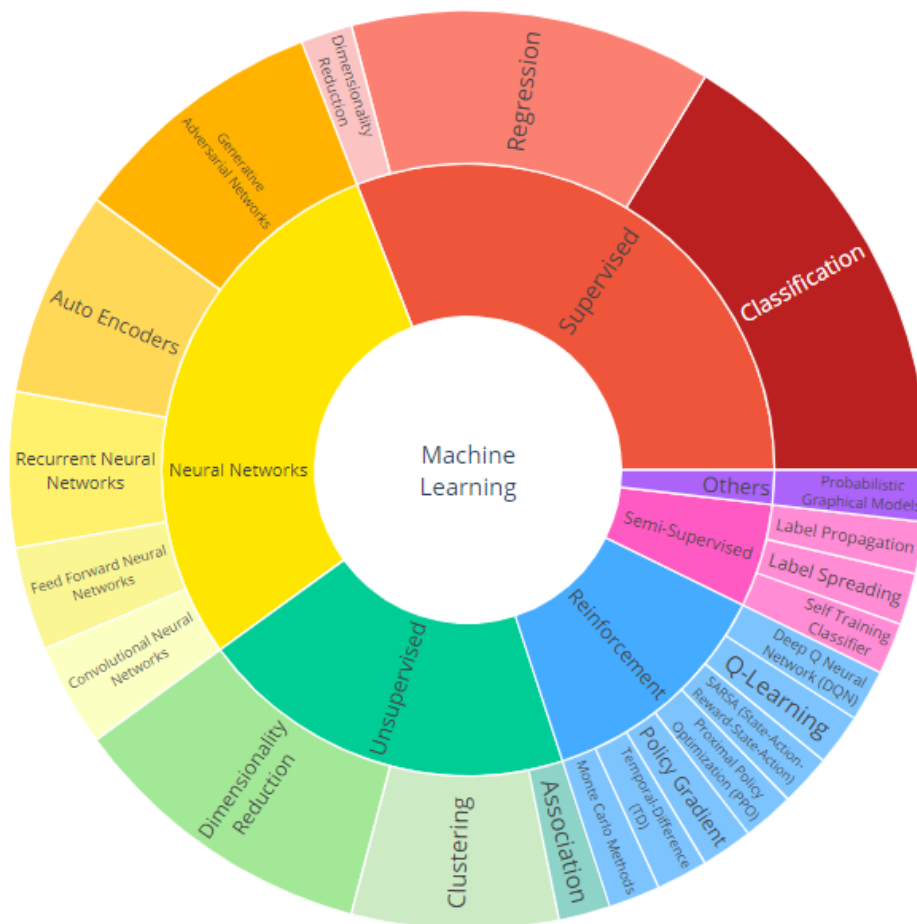
۵-۱-۱ مدل RNN

مدل سازی و پیش بینی داده های متوالی نیازمند رویکردی متفاوت از رگرسیون یا طبقه بندی استاندارد است. خوشبختانه، نوع خاصی از شبکه های عصبی به نام شبکه های عصبی بازگشتی (RNN) به طور خاص برای این منظور طراحی شده اند.

در این مقاله به ساختار RNN ها می پردازم و یک مثال کامل از نحوه ساخت یک RNN ساده با استفاده از Keras و Tensorflow در پایتون به شما ارائه می کنم.

۵-۲ نگاهی به جهان یادگیری ماشین

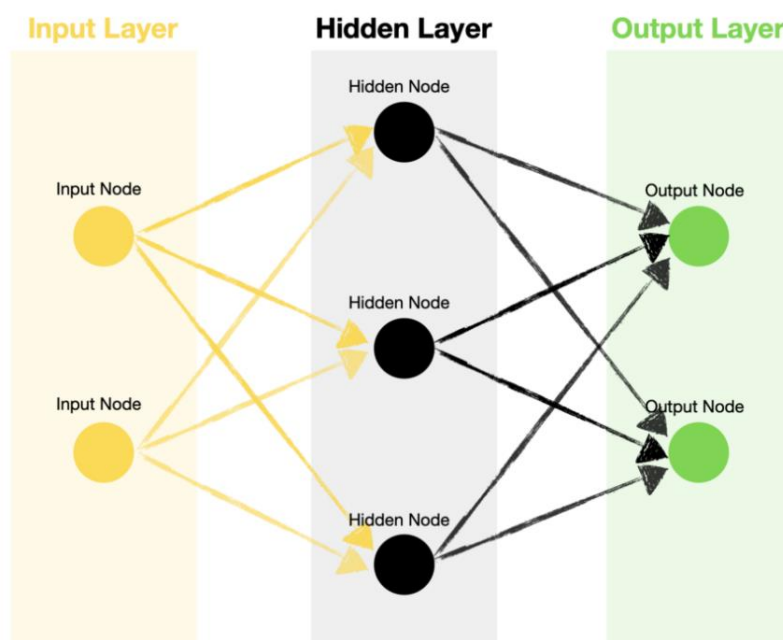
در حالی که شبکه های عصبی اغلب به شیوه ای تحت نظارت با داده های آموزشی برچسب گذاری شده استفاده می شوند، من احساس کردم که رویکرد منحصر به فرد آنها به یادگیری ماشینی مستحق یک دسته جداگانه است. شبکه های عصبی مکرر زیرشاخه خود را دارند که از RNN های ساده، LSTM (حافظه کوتاه مدت بلندمدت) و GRU (واحد بازگشتی دردار) تشکیل شده است.



تصویر شماره ۲۲

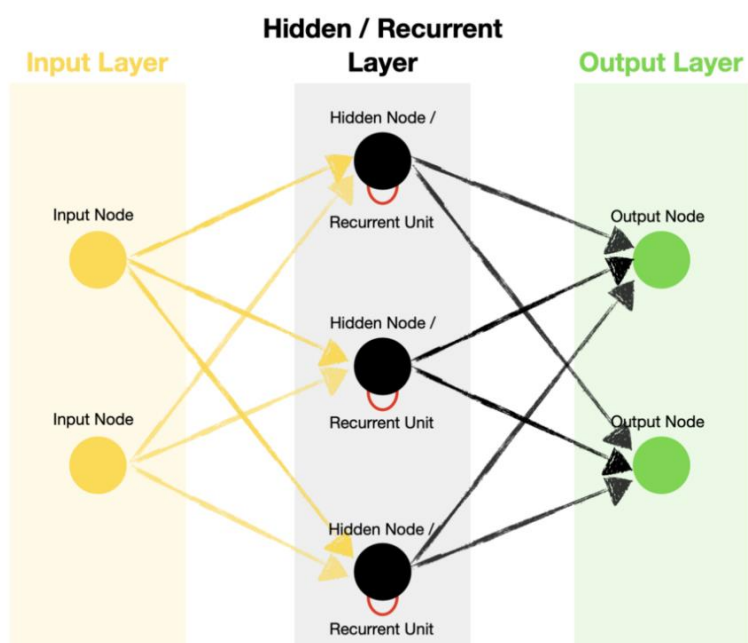
۳-۵ ساختار شبکه های عصبی بازگشتی (RNN)

ابتدا، بیاپید به خود یادآوری کنیم که یک شبکه عصبی فید فوروارد معمولی چگونه است. توجه داشته باشید که می تواند شامل هر تعداد گره ورودی، گره پنهان و گره خروجی باشد. ساختار ۲-۳-۲ زیر صرفاً برای مثال است.



تصویر شماره ۲۳

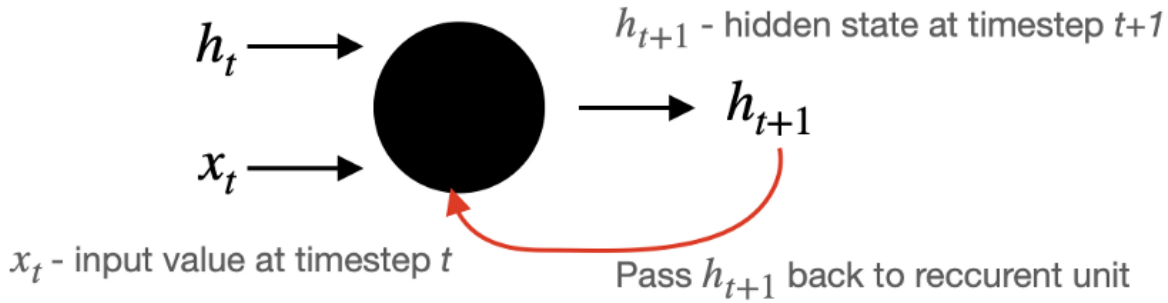
در مرحله بعد، اگر به RNN نگاه کنیم، متوجه تفاوت جزئی می شویم. واحدهای پنهان در داخل RNN دارای یک حلقه بازخورد داخلی هستند که اطلاعات را قادر می سازد چندین بار به یک گره بازگردانده شود. این واحدهای پنهان معمولاً واحدهای مکرر نامیده می شوند.



تصویر شماره ۲۴

تصویر زیر را ببینید که حلقه بازخورد داخل واحد برگشتی را نشان می دهد:

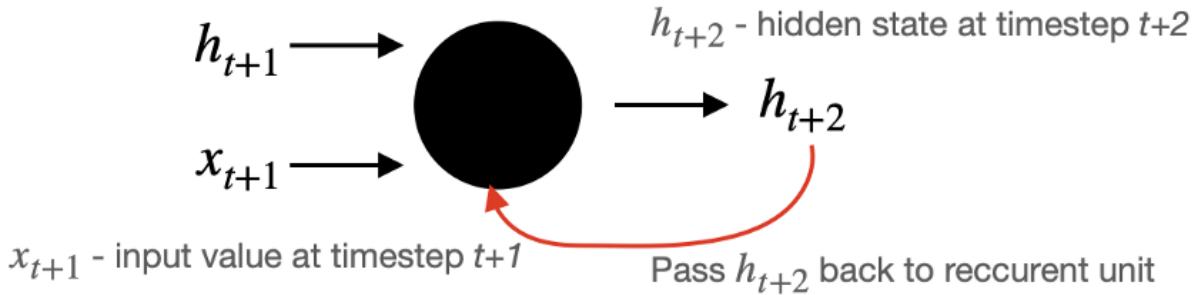
h_t - hidden state at timestep t



تصویر شماره ۲۵

توجه داشته باشید که در مرحله زمانی اولیه، حالت پنهان h_0 به ۰ مقداری می شود. سپس خروجی (یک حالت پنهان h در $t+1$) به یک واحد بازگشتی ارسال می شود و دوباره همراه با ورودی زیر پردازش می شود:

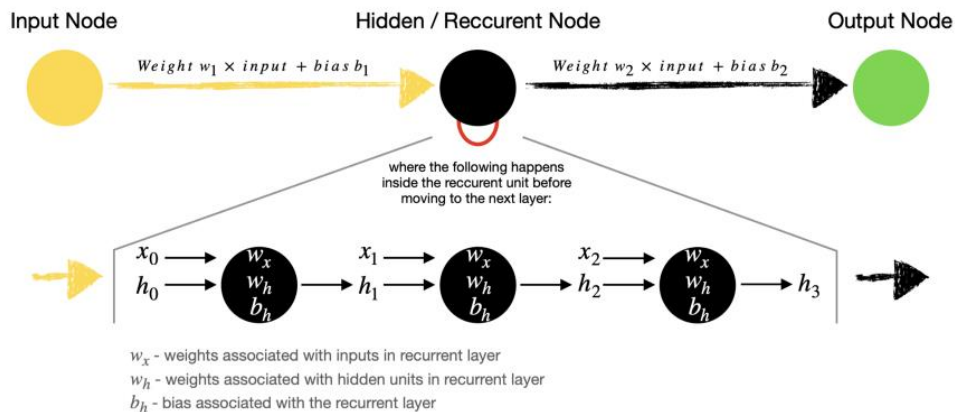
h_{t+1} - hidden state at timestep $t+1$



تصویر شماره ۲۶

این فرآیند تا رسیدن به تعداد زمان مشخص شده تکرار می شود.

بیایید همه آن را به هم گره بزنیم و ببینیم یک RNN ساده با یک ورودی، یک گره پنهان (شامل سه مرحله زمانی) و یک خروجی چگونه خواهد بود.



تصویر شماره ۲۷

۴-۵ توضیح کد model.py

در این قسمت به توضیح کد model.py و معماری ارائه شده برای مدل بازگشتی میپردازیم.

در تصویر زیر لایه های این مدل و اطلاعات آنها آورده شده است.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 45)	10620
dense (Dense)	(None, 32)	1472
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 2)	66
Total params: 12,158		
Trainable params: 12,158		
Non-trainable params: 0		

تصویر شماره ۲۸

ابتدا یک لایه Lstm با ۴۵ واحد در ابتدای مدل قرار میگیرد. سپس خروجی این لایه به یک لایه fully connected با ۳۲ نرون داده میشود که این لایه با یک لایه dropout دنبال میشود تا از overfit شدن مدل جلوگیری کند. در انتها نیز لایه ی خروجی با دو نرون وجود دارد. معماری این مدل برای جلوگیری از overfit شدن مدل، ساده تر شده است که در قسمت های بعدی به این موضوع میپردازیم.

در ادامه توضیحات کد model.py را خواهیم داشت.

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import pandas as pd
```


ابتدا کتابخانه های مورد استفاده در این کد را اضافه میکنیم. کتابخانه ی json که برای خواندن دیتاست ها استفاده میشود. کتابخانه keras که برای ایجاد مدل ها و آموزش آنها مورد استفاده قرار میگیرد.

```
# RNN-LSTM model
input_shape = (X_train.shape[1], X_train.shape[2]) # ( # of segments, # of coeffs
learning_rate = 0.0001
```

در این قسمت از کد پارامتر های مورد استفاده برای مدل را مقدار دهی میکنیم. ابعاد ورودی شبکه عصبی که در متغیر input_shape ذخیره شده است، با استفاده از مقادیر shape که ابعاد یک ماتریس از کتابخانه numpy را در خود نگه میدارد، مقدار دهی شده است. برای شبکه های عصبی بازگشتی ابعاد ورودی باید به صورت زیر باشد:

[number of time steps, number of features]

که در مسئله ی ما تعداد time step ها همان تعداد قسمت های هر سیگنال صوتی میباشد و تعداد ویژگی ها نیز همان تعداد ضرایب استخراج شده از هر سیگنال میباشد.

```
model = keras.Sequential()
```

ابتدا با استفاده از تابع Sequential از کتابخانه keras یک مدل پایه ایجاد میکنیم. این مدل در ابتدا هیچ لایه ای ندارد اما با استفاده از تابع add از همین کتابخانه میتوان به آن به ترتیب لایه اضافه کرد.

```
model.add(keras.layers.LSTM(45, input_shape=input_shape))
```

با استفاده از تابع add یک لایه ی LSTM با ۴۵ واحد به مدل اضافه میکنیم. برای ایجاد این لایه از ماژول layers در کتابخانه keras استفاده شده است که در این ماژول انواع لایه های شبکه های عصبی وجود دارد. این لایه بصورت many to one میباشد چراکه تنها خروجی آخرین واحد از این لایه به لایه بعدی داده میشود. این لایه یک ورودی دیگر علاوه بر تعداد واحد ها به نام input_shape دارد که ابعاد ورودی لایه را مشخص میکند (به این خاطر که این لایه اولین لایه در مدل میباشد نیاز هست تا این مقدار برای آن مشخص شود. ابعاد ورودی برای لایه های بعدی با توجه به ابعاد خروجی لایه قبلیشان، مشخص میشود)

```
model.add(keras.layers.Dense(32,activation='relu'))
model.add(keras.layers.Dropout(0.35))
```

بعد از لایه LSTM یک لایه fully connected با ۳۲ نرون و به دنبال آن یک لایه dropout اضافه شده است. تابع فعالسازی برای لایه fully connected برابر با relu قرار داده شده و پارامتر ورودی برای لایه dropout که احتمال غیرفعالسازی هر نرون در هر تکرار را نشان میدهد برابر با ۰,۳۵ قرار داده شده است.

```
#softmax
model.add(keras.layers.Dense(2,activation='softmax'))
```

در انتها لایه خروجی که یک لایه fully connected با ۲ نرون میباشد به مدل اضافه شده است. تابع فعالسازی این لایه به دلیل اینکه یک احتمال برای هر کلاس میخواهیم، softmax در نظر گرفته شده است.

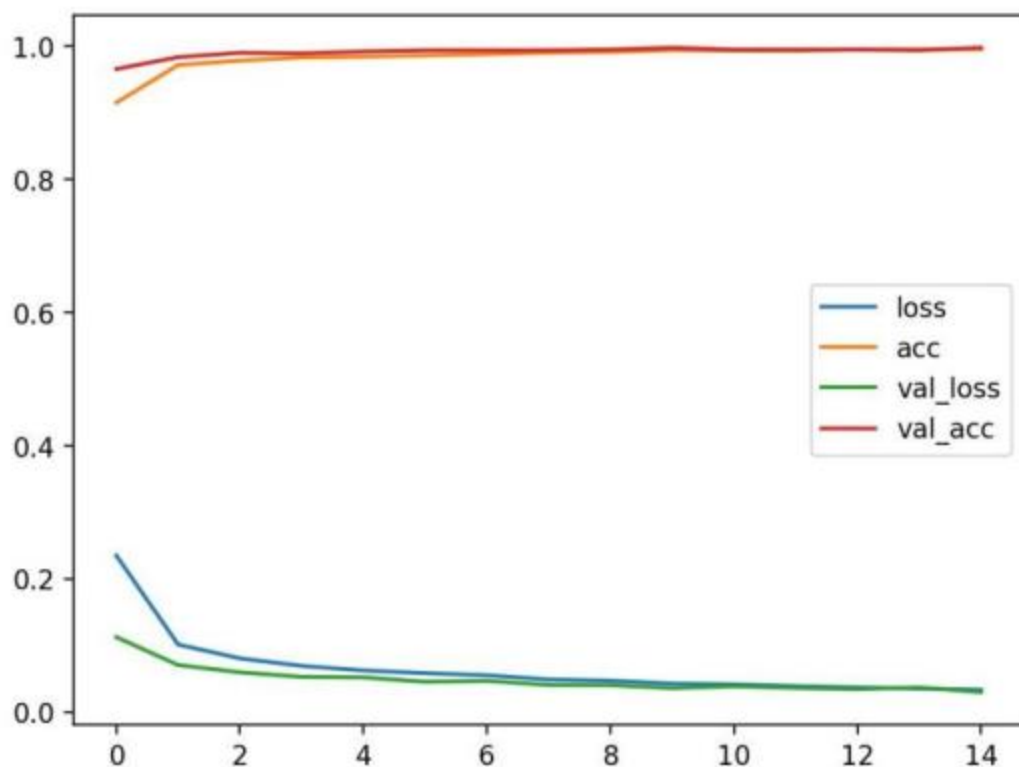
```
#compile
optimiser = keras.optimizers.Adam(learning_rate)
model.compile(optimizer = optimiser, loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

در این قسمت الگوریتم بهینه سازی مدل برابر با الگوریتم Adam optimizer قرار داده شده است. برای نهایی سازی مدل و مقداردهی پارامترهای آن از تابع compile مدل میتوان استفاده کرد. برای این تابع الگوریتم بهینه سازی و تابع ضرر مشخص شده است. تابع ضرر برای این مسئله sparse categorical cross entropy در نظر گرفته شده که همان تابع cross entropy میباشد که در مسائل کلاس بندی کاربرد دارد.

```
***** train the model
model.fit(X_train, Y_train, epochs = 15, batch_size = 32, validation_data = (X_val,Y_val))

loss = pd.DataFrame(model.history.history)
loss.plot()
```

سپس مدل با تابع fit آموزش داده میشود. ورودی اول و دوم این تابع دیتاست های آموزش میباشد. ورودی epoch تعداد تکرار بر روی کل دیتاست را مشخص میکند که برابر با ۱۵ قرار داده شده است. ورودی batch_size تعداد اعضای هر mini batch در الگوریتم mini batch stochastic gradient descent (به صورت پیش فرض از این الگوریتم برای بهینه سازی در مدل های Sequential کتابخانه keras استفاده میشود) را مشخص میکند که برابر با ۳۲ قرار داده شده است. در آخرین ورودی این تابع نیز دیتاست های validation مقداردهی شده اند. در خط بعدی مقدار خطا در هر تکرار را از مدل استخراج میکنیم و آن را بصورت یک جدول با استفاده از تابع DataFrame از کتابخانه pandas ذخیره میکنیم. در خط بعد با استفاده از تابع plot موجود برای DataFrame ها، نمودار خطا در طول زمان را رسم میکنیم.



تصویر شماره ۲۹

در نمودار بالا میتوان تغییرات خطا در هر تکرار را برای داده های آموزش و داده های validation مشاهده کرد.

```
#save model
model.save(path+'//rnnmodel.h5')
```

در نهایت مدل را با فرمت h5 با استفاده از تابع save ذخیره میکنیم.

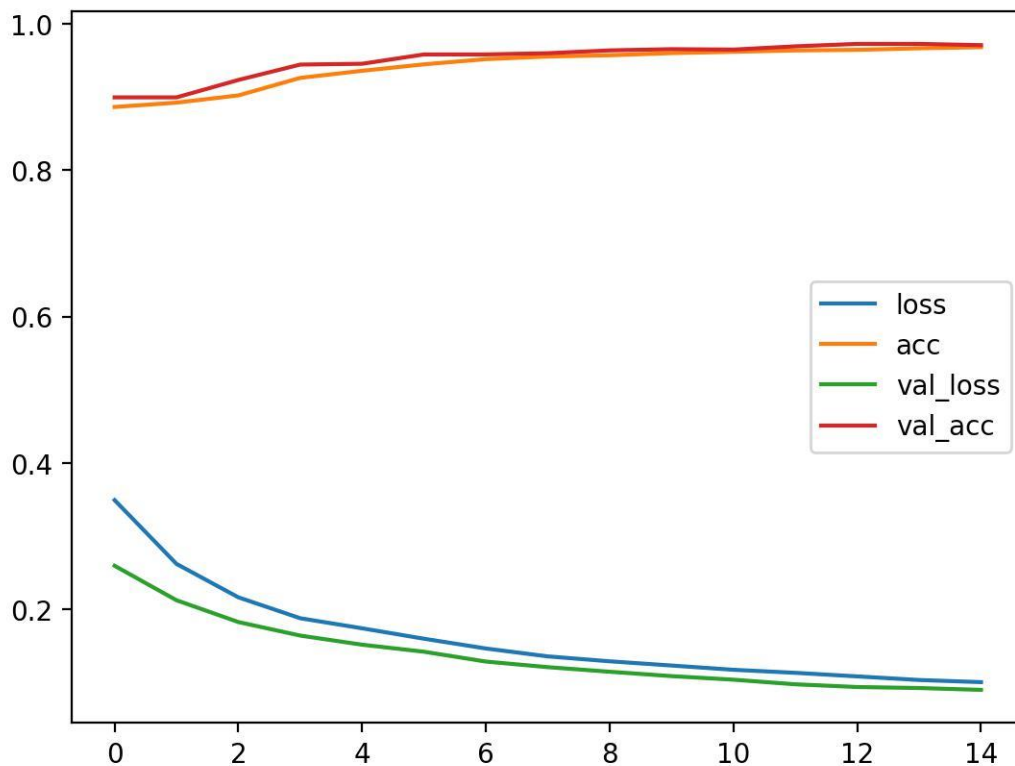
نتایج بدست آمده از آموزش شبکه عصبی کانوولوشنال

```
test error: 0.07646160735899671
test accuracy: 0.97356606
```

در تصویر زیر نیز عملکرد مدل را مشاهده می کنید:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1803
1	0.89	0.84	0.87	202
accuracy			0.97	2005
macro avg	0.94	0.91	0.93	2005
weighted avg	0.97	0.97	0.97	2005

همانطور که در تصویر بالا مشاهده می شود، دقت مدل برابر با ۰,۹۸ می باشد که این عدد به علت نوع مسئله می باشد و این مدل از مشکلات **overfitting** به اندازه مدل قبلی رنج نمی برد و برای داده های قبلی به خوبی عمل می کند.



۵-۵ راه حل های استفاده شده برای overfitting

همانطور که پیشتر اشاره شد مدل اول که پیاده سازی شد مشکل **overfitting** داشت به این معنا که تنها داده های مجموعه دادگان آموزش را به خوبی یاد گرفته بود و به عبارتی آنها را حفظ کرده بود. برای حل این مشکل چندین اقدام انجام گرفت. اولین اقدام اضافه کردن دیتا به دیتاست بود. برای اینکار کد **record.py** پیاده سازی شد که در قسمت های قبلی توضیح داده شد. از دیگر اقدامات میتوان به ساده کردن معماری مدل اشاره کرد که در قسمت پیاده سازی مدل توضیح داده شد. برای اینکار تعداد لایه های

شبکه کم شدند و تعداد فیلترهای هر لایه نیز کاهش یافتند. از دیگر اقدامات بالا بردن احتمال dropout در شبکه عصبی بود. این لایه باعث خاموش شدن نرون‌ها در هر epoch میشود که این امر از تغییر وزن‌های این نرون‌ها جلوگیری میکند و از حفظ کردن داده‌ها توسط مدل و overfit شدن جلوگیری میکند. از دیگر اقدامات میتوان به کم کردن تعداد epochها اشاره کرد. این امر باعث کمتر دیده شدن داده‌ها توسط شبکه میشود که باعث جلوگیری از حفظ کردن آنها میشود. یکی دیگر از مشکلات مدل تشخیص نویز و صدای محیط به عنوان کلمه کلیدی بود که برای جلوگیری از این موضوع صوت‌هایی با محتوای صدای محیط و نویز به صورت دستی ضبط و به مجموعه دادگان اضافه شدند.

۶-۵ مقایسه عملکرد مدل‌ها

در این قسمت به مقایسه عملکرد مدل‌های پیاده‌سازی شده در این پروژه می‌پردازیم. در جدول زیر میتوان معیارهای ارزیابی مختلف را برای هر کدام از این مدل‌ها مشاهده کرد.

مدل	خطای تست	دقت تست	کلاس	precision	Recall	F1-score
شبکه عصبی کانولوشنال اولیه	0.043	0.98	کلمه کلیدی (کلاس 1)	0.95	0.93	0.94
			کلاس 0	0.99	1.00	0.99
شبکه عصبی کانولوشنال بهبود یافته	0.038	0.99	کلمه کلیدی (کلاس 1)	0.98	0.96	0.97
			کلاس 0	1.00	1.00	1.00
شبکه عصبی بازگشتی	0.076	0.97	کلمه کلیدی (کلاس 1)	0.89	0.84	0.87
			کلاس 0	0.98	0.99	0.99

فصل ششم

۱-۶ خلاصه

به طور کلی این پروژه به دو بخش یادگیری ماشین و طراحی و پیاده سازی برنامه دسکتاپ تقسیم شده بود که بخش طراحی و پیاده سازی برنامه دسکتاپ پروژه بر عهده اینجانب بود.

کارهای انجام شده در بخش پیاده سازی برنامه دسکتاپ به همان صورت که در فصول قبل آورده شده بود می باشد و شامل مراحل مختلف از جمله پیاده سازی کد `app.py` برای طراحی رابط کاربری گرافیکی در فصل ۴، کد `real_time.py` برای گرفتن صوت از ورودی میکروفون در حین اجرا و رسم نمودار و سپس فرستادن یک یک ثانیه ی صوت به `model.py` در فصل ۴ و کد `model.py` برای ساخت مدل ها به منظور `train` کردن آنها که در فصل ۵ آورده شده بود.

۲-۶ نتیجه گیری

شکی نیست که دستیارهای صوتی شاهکار بزرگی از نبوغ بشر هستند و خواهند شد و در حال حاضر به شکل یا شکلی وارد زندگی ما شده اند. با عرضه نهایی اینترنت 5G و بهبود دستیارهای صوتی یادگیری ماشینی ممکن است خود را به عنوان ابزاری تبدیل کنند که نمی توانیم بدون آن زندگی کنیم.

با این حال، قبل از رسیدن به آن مرحله، موانعی برای عبور وجود دارد که شامل سرمایه گذاری سنگین، بهبود فناوری و اعتماد مصرف کنندگان به این است که این دستگاه که در زندگی آنها است خطری برای حریم خصوصی آنها ندارد.

مراجع

[1] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuoyiin Chang, Tara Sainath. Deep Learning for Audio Signal Processing.

[2] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, Koray kavukeoglu. Efficient Neural Audio Synthesis

[3] Yefei Chen, Heinrich Dinkel, Mengyue Wu, Kai Yu. Voice activity detection in the wild via weakly supervised sound event detection

[4] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, Zhenyao Zhu. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin

[5] Yun-Chia Liang, Iven Wijaya, Ming-Tao Yang, Josue Rodolfo Cuevas Juarez, Hou-Tai Chang. Deep Learning for Infant Cry Recognition

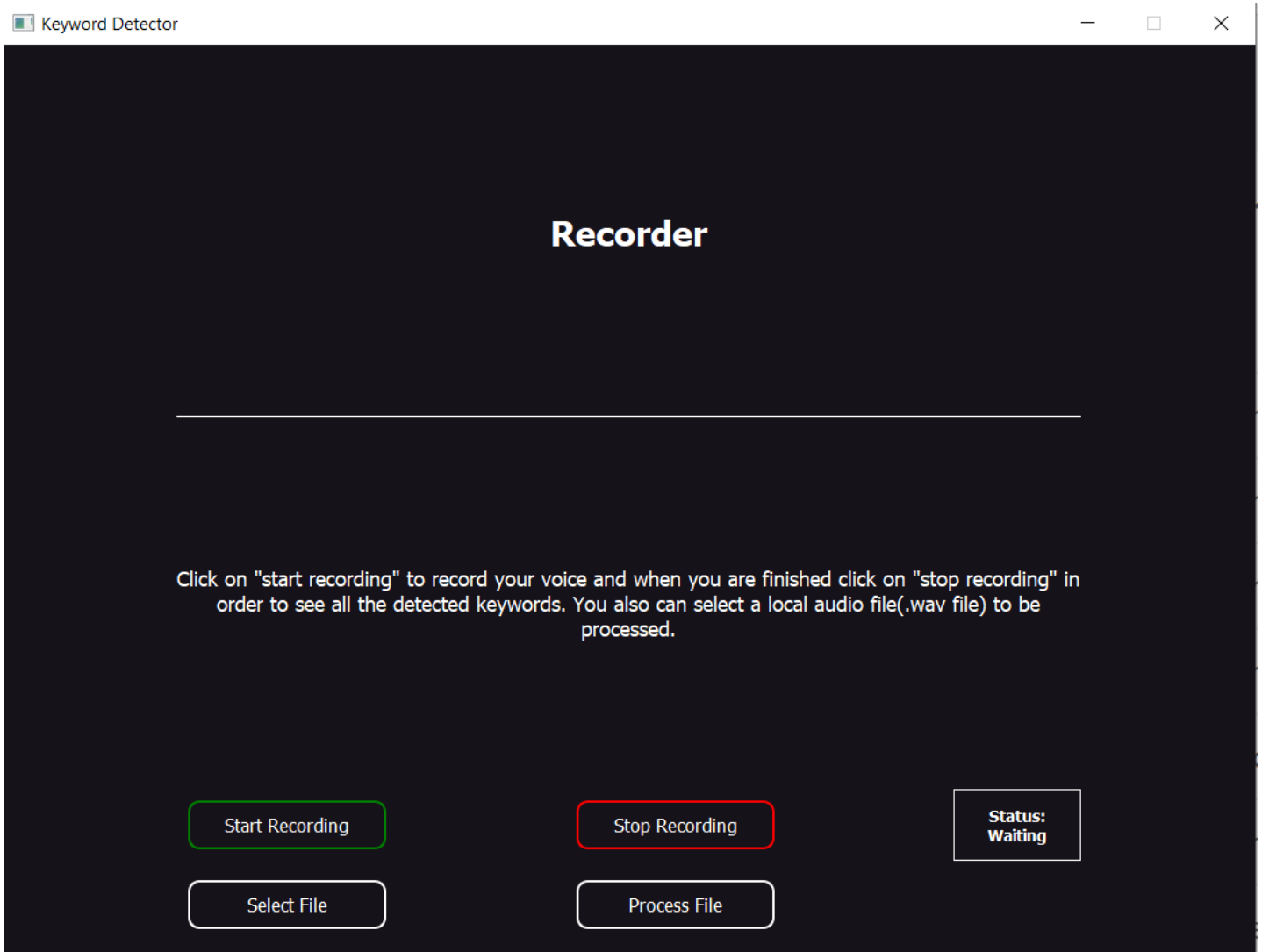
[6] Francois Chollet. Deep Learning with Python

[7] ایمان عباسپور کازرونی، حسن مسلمی، فاطمه حورعلی : تشخیص هوشمند اختلال گفتاری با استفاده از سیستم پردازش سیگنال صوت

پیوست

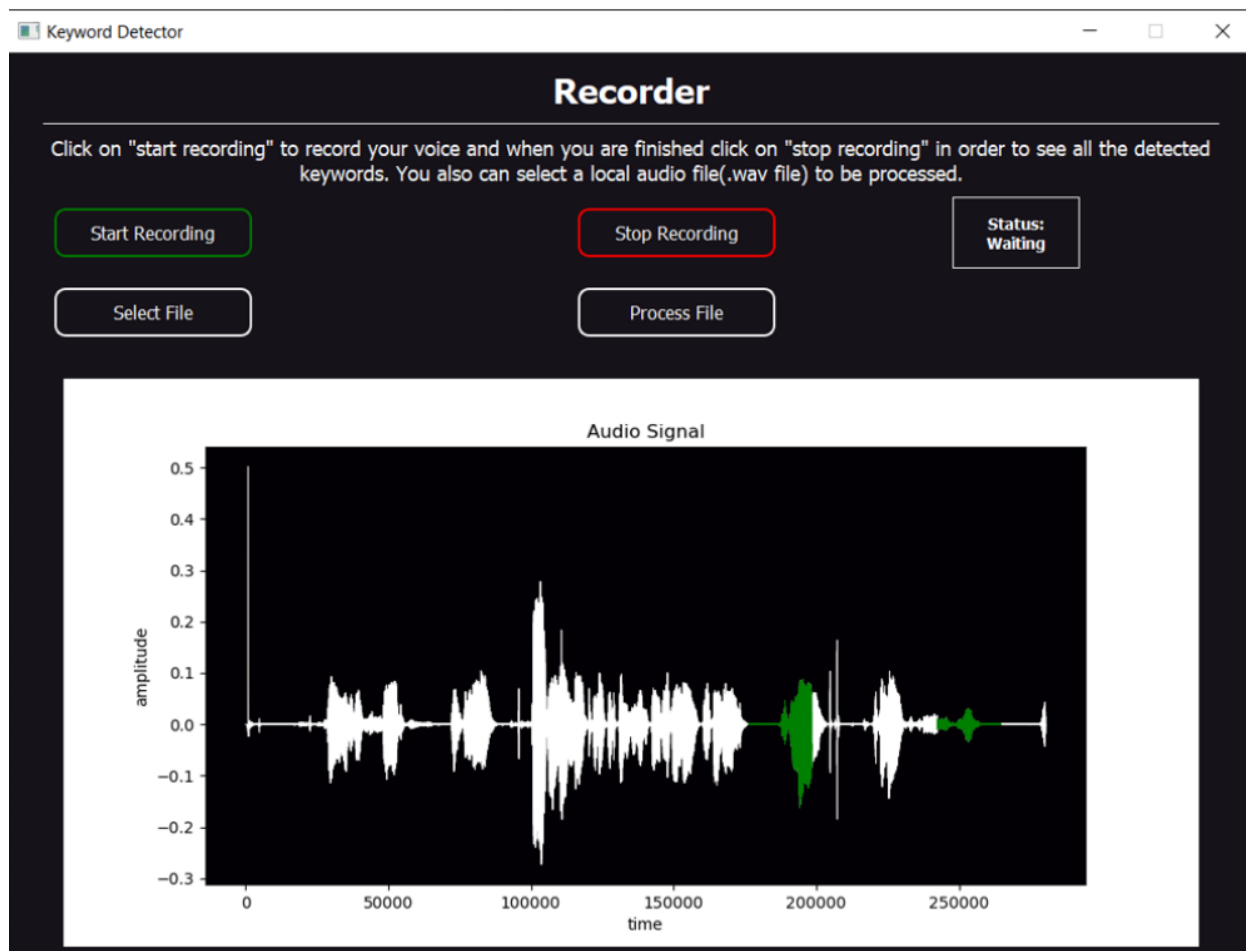
نحوه کار با برنامه دسکتاپ

ابتدا کد app.py را اجرا میکنیم تا رابط کاربری گرافیکی نمایش داده شود.



تصویر شماره ۳۰

برای استفاده از قابلیت تشخیص صوت در صوت ضبط شده ابتدا بر روی دکمه Start Recording کلیک میکنیم و زمانی که status به Listening تبدیل شد به این معنی است که برنامه در حال دریافت صدا از میکروفون میباشد. هر زمان که خواستیم تا ضبط متوقف شود بر روی Stop Recording کلیک میکنیم و منتظر خروجی برنامه میمانیم. خروجی برنامه به صورت تصویر زیر به نمایش گذاشته میشود:

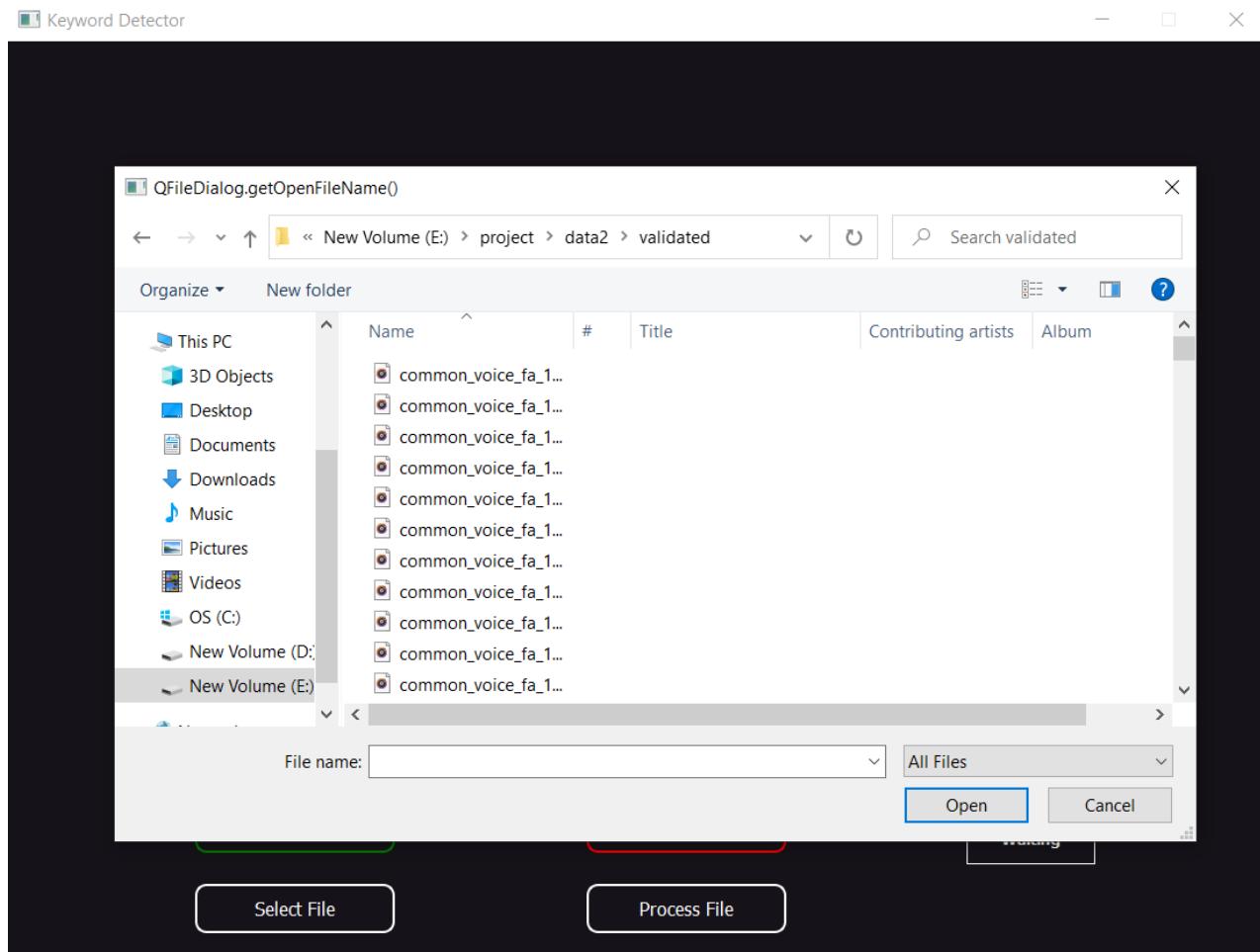


تصویر شماره ۳۱

که در آن سیگنال صوت ضبط شده نمایش داده میشود و قسمت های سبز نشان دهنده ی بخش هایی از صوت است که برنامه کلمه کلیدی را تشخیص داده است. همچنین در فولدری با نام detected فایل های صوتی ۱ ثانیه ای از بخش هایی که کلمه کلیدی تشخیص داده شده قرار میگیرد.

برای استفاده از قابلیت انجام پردازش بر روی یک فایل داخل سیستم خود، روی Select File کلیک میکنیم و فایل مورد نظر خود را انتخاب میکنیم. زمانی که status به Processing تبدیل شد بر روی Process File کلیک کرده تا پردازش روی فایل صورت بگیرد و خروجی نمایش داده شود. خروجی این بخش هم مانند بخش قبل میباشد.

در تصویر زیر قسمت انتخاب فایل برای این قابلیت را مشاهده میکنید:



تصویر شماره ۳۱



University of Tehran

College of Farabi

Faculty of Engineering

Department of Computer Engineering

Implementation of a Graphical Desktop App for Persian Speech Recognition System

By:

Seyyed Sina Mousavi Sabet

Under Supervision of:

Dr. Zahra Movahedi

A Project Report as a Requirement for the Degree of Bachelor of Science in Computer Engineering

September 2022