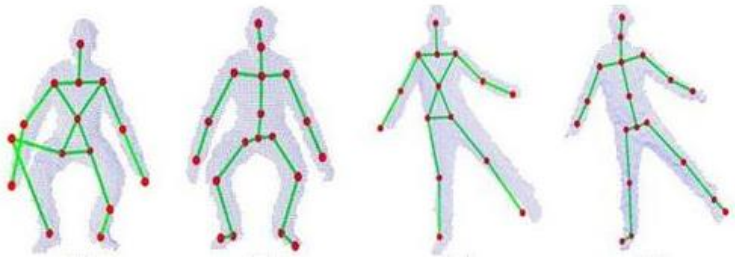
بسمه تعالی

دانشگاه صنعتی شریف

دانشکده مهندسی برق

گروه سیستم‌های دیجیتال



آزمایشگاه یادگیری و بینایی ماشین

دستور کار آزمایش نهم: شبکه های عصبی

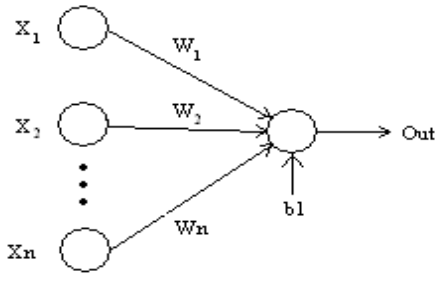
زمان لازم برای انجام آزمایش: حداکثر دو جلسه

آزمایش نهم: شبکه های عصبی

می توان گفت اولین تلاش ها برای توضیح و بررسی رابطه و وابستگی ساختارهای عصبی و سیگنال های الکتریکی به حدود سال ۱۸۸۰ باز می گردد. دانشمندان زیادی تلاش می کردند تا مدلی برای عملکرد مغر انسان ارائه کنند. مشکلی که بسیاری از مدل های اولیه داشتند این بود که نمی توانستند قابلیت یادگیری که در ذهن انسان وجود دارد را داشته باشند. در این مسیر دیدگاه های متعدی بین دانشمندان به وجود آمد. پس از پیشرفت هایی در این حوزه، انسان به ابزاری دست یافته بود که می توانست برای پردازش بسیاری از مسائل از آن استفاده کند. می توان گفت در دهه ۸۰ میلادی مسیر پژوهش ها در این زمینه به دو دسته بزرگ تقسیم شد: گروهی بر هدف اولیه پایبند بودند و سعی بر این داشتند مدل ها را طوری گسترش دهند که بتوانند به درکی از عملکرد مغز انسان برسند. گروهی دیگر نیز به ابزار خوبی برای پردازش و حل مسائل مختلف دست یافته بودند که غالب آن ها را مهندسان تشکیل می دادند. این دسته سعی داشتند دقت و سرعت مدل ها را برای پردازش تقویت کنند و خیلی توجهی به این نداشتند که بتوانند مدلی برای عملکرد مغز انسان بیابند.

پرسپترون نوعی از شبکه های عصبی مصنوعی است که در دهه ۵۰ میلادی به وسیله فرانک روزنبلات تدوین شد. ریشه کار های روزنبلات را می توان در مدلی که مک کلاوچ(McCilloch) و پیتز(Pitts) ارائه کرده بودند جستجو کرد.

اما پرسپترون چگونه کار می کند؟ بر اساس یافته های بایولوژیکی مدلی استخراج شد که ورودی و خروجی دو وضعیتی(باینری) داشته باشد. بر این اساس به توصیف زیر می رسیم:



Perceptron Model

$$output = \begin{cases} 1 & if \sum_j w_j x_j + b > 0 \\ 0 & otherwise \end{cases}$$

w_j ها وزن ها با مقادیر حقیقی هستند. در رابطه بالا b نشان‌دهنده بایاس است که وظیفه آن جابجا کردن مرز تصمیم‌گیری از مبدأ است و مقدار آن به ورودی‌ها(x_j) بستگی ندارد.

این یک مدل ساده ریاضی است که با تغییر وزن ها می توان به تصمیم گیری های متفاوت رسید.

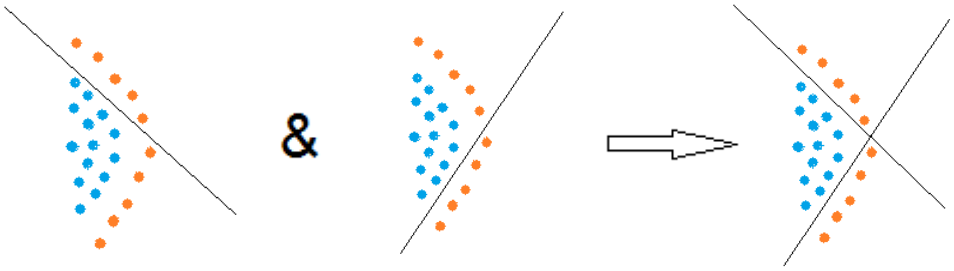
(امتیازی)AND GATE با دو ورودی را با پرسپترون بسازید.

کاری که یک شبکه پرسپترون تک لایه می کند در واقع این است که در یک فضای n بعدی یک ابر صفحه(Hyperplane) n بعدی در نظر می گیرد. اگر داده ای بالای این ابر صفحه بود لیبل آن را ۱ در نظر می گیرد و در غیر این صورت لیبل آن را ۰ می گیرد.

از همین جا مشخص می شود اگر چنین شبکه ای قابلیت یادگیری داشته باشد فقط پترن هایی را می تواند یاد بگیرد که خطی تفکیک پذیر باشند. برای مثال پترن زیر را نمی توان یادگرفت:

از جمله کار هایی که می توان انجام داد تا این مشکل را حل کرد این است که داده ها را به فضایی بالاتر نگاشت کنیم طوری که در آن فضا خطی تفکیک پذیر شوند.(تعداد ورودی ها زیاد می شوند.)

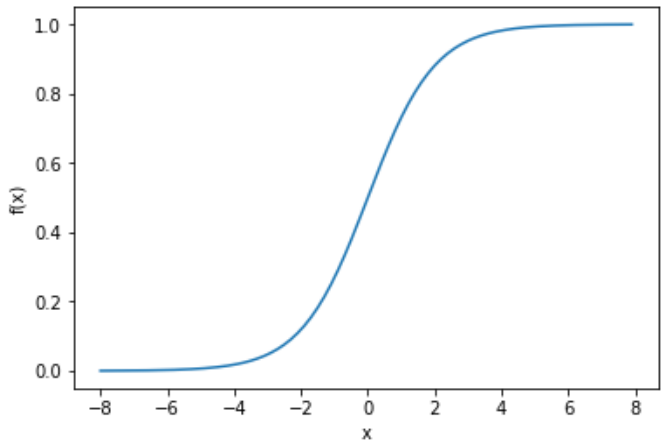
کار دیگری که می توان انجام داد این است که از شبکه ای با بیش از یک لایه استفاده کنیم.



در واقع سعی می کنیم با AND چند ابر صفحه به فضای دلخواه خود برسیم.

تولد شبکه های عصبی از جایی بود که دانشمندان تصمیم گرفتند مدلی برای پروسه یادگیری مغز انسان طراحی کنند. منطقی به نظر می رسد اگر بگوییم رابطه ورودی و خروجی یک نورون، به صورت رابطه پله ای که در بالا (perceptron)توصیف شد نمی تواند باشد. در واقع بررسی های بایولوژیکی نشان می دهند وقتی که غلظت یون ها اطراف شاخک های گیرنده یک نورون افزایش می یابد، آن نورون فعال می شود. پس به نظر می رسد باید برای بهتر توصیف کردن رابطه ورودی و خروجی نورون، به دنبال تابعی پیوسته باشید. به تابعی که رابطه ورودی و خروجی نورون را توصیف می کند، تابع فعال سازی می گویند.(Activation Function). تابع sigmoid یکی از معروف ترین Activation Function ها می باشد که به صورت زیر تعریف می شود:

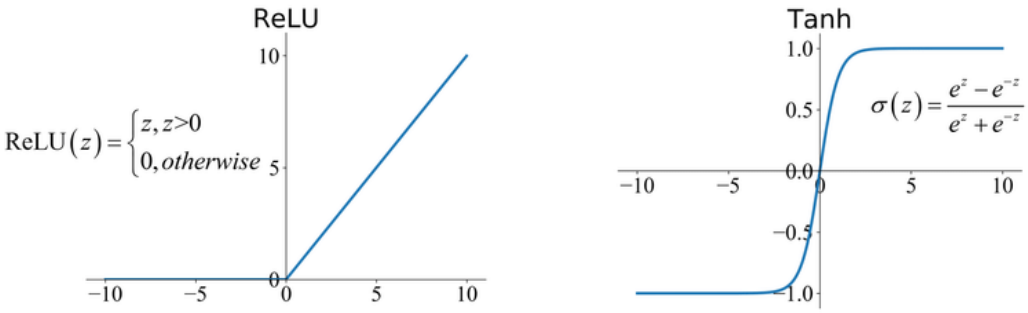
$$f(x) = \frac{1}{1 + e^{-x}}$$



Sigmoid Activation Function

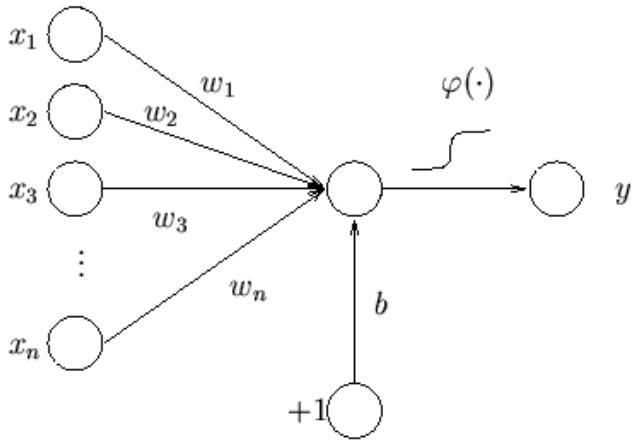
از جمله ضعف هایی که تابع فعال سازی sigmoid دارد این است مشتق آن وقتی به x به سمت بی نهایت می رود بسیار کوچک می شود. این مساله همگرایی شبکه در آموزش را به شدت کند می کند. (در صفحات بعد درباره نحوه آموزش شبکه و تاثیر گرادیان – مشتق – تابع فعال ساز خواهید خواند.)

توابع فعالسازی دیگری نیز وجود دارند: hyperbolic tangent (tanh) و Rectifier Linear Unit (ReLU) از معروف ترین این توابع هستند.



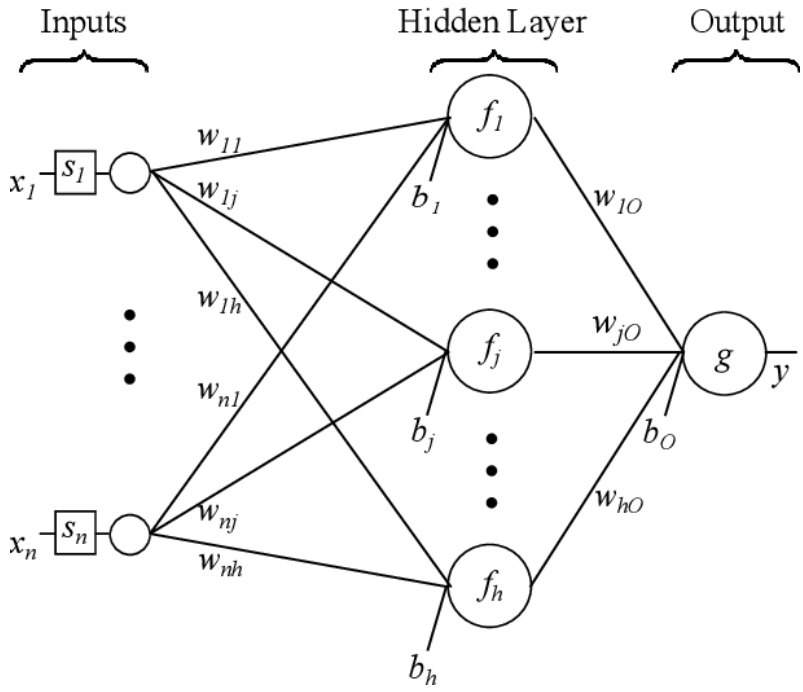
ReLU and tanh Activation Function

در نهایت با اعمال تابع فعال سازی ساختار یک نورون به صورت زیر نهایی میشود:



Single Layer Perceptron Structure

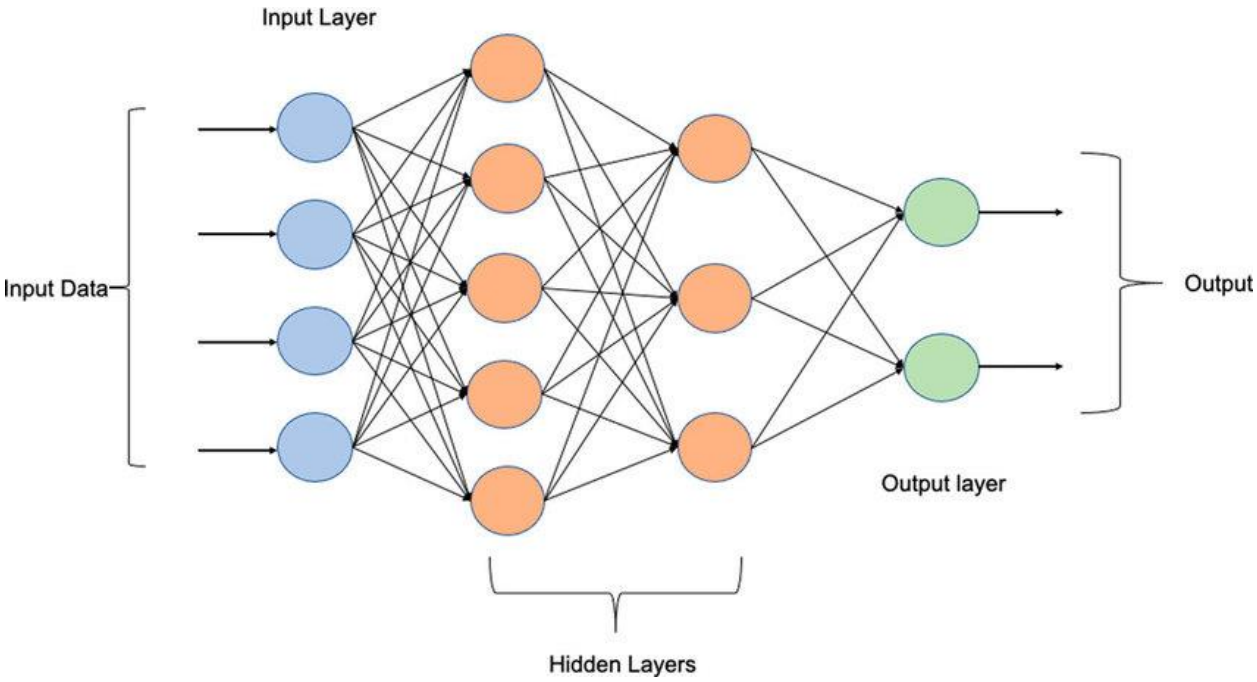
با اتصال نورون ها به یک دیگر به شبکه های عصبی می رسیم. شبکه های عصبی اغلب از سه لایه تشکیل شده اند: $output\ layer$ و $hidden\ layer$ و $input\ layer$.



Multilayer Perceptron (MLP) with single hidden layer

شکل بالا شبکه ای با یک $hidden\ layer$ را نشان میدهد که به آن $Multilayer\ perceptron$ میگویند. هر ورودی با وزن های مشخصی به تمام نورون های لایه مخفی متصل میشود. همچنین هر نورون لایه مخفی مقدار بایاس مشخصی دارد که با سایر نورون ها متفاوت است. خروجی تمام نورون های لایه مخفی از تابع فعالسازی عبور میکنند. این خروجی ها به منزله ورودی های لایه خروجی هستند. این مقادیر به مانند لایه ورودی با وزن های مختلف و بایاس به نورون های لایه آخر میروند. در لایه آخر در نهایت با عبور از تابع فعال سازی، خروجی شبکه بدست می آید.

برای گسترش یک شبکه میتوان تعداد نورون ها در لایه های مختلف را افزایش داد یا تعداد لایه ها را افزایش داد.

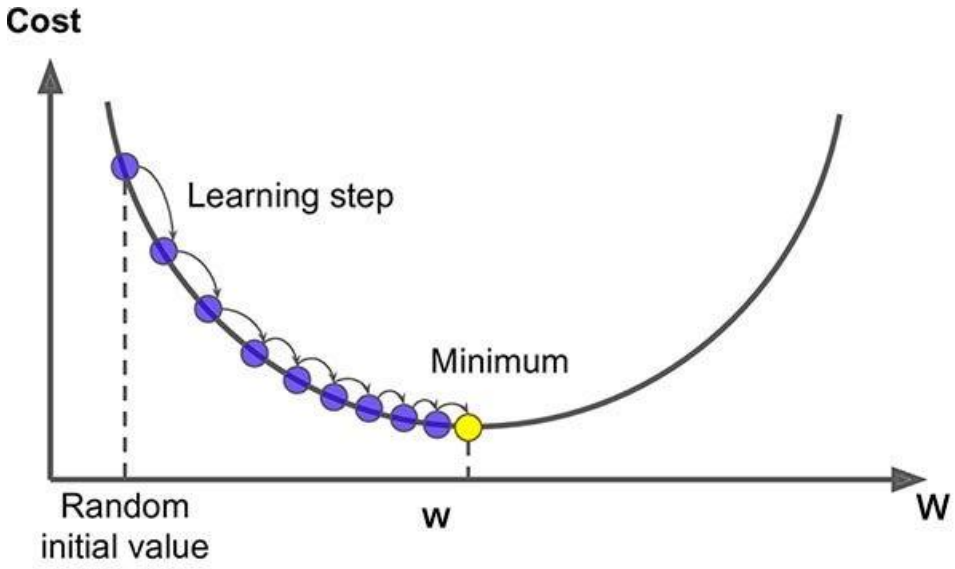


Multilayer Perceptron (MLP) with 2 hidden layers

در تصویر بالا شبکه ای با ۲ لایه مخفی و تعداد مختلف نورون در لایه های مختلف دیده میشود.

پس از معرفی ساختار کلی شبکه های عصبی سوالی که مطرح میشود این است: مقادیر مناسب ضرایب یک شبکه عصبی چگونه تعیین میشوند؟ فرآیند آموزش چگونه است؟

در ابتدای کار و با وجود مقادیر اولیه و رندم وزن ها، شبکه برای تمام داده های آموزشی ورودی مقادیر احتمال تعلق به هر کلاس را تخمین میزند (اگر تسک مورد نظر طبقه بندی باشد.) با مقایسه این مقادیر و مقادیر مورد انتظار، میتوان میزان (تابع هزینه) را مشخص کرد. (مثلا Mean Square Error – MSE برای رگرشن یا Cross Entropy Loss – CE برای طبقه بندی چند کلاسه) حال وزن های شبکه باید به گونه ای تغییر کنند که در جهت حداقل کردن این خطا پیش برویم. برای حداقل کردن تابع هزینه از الگوریتم بهینه سازی gradient descent استفاده میشود. این الگوریتم از نوع الگوریتم های تکرار شونده مرتبه اول است. بدین معنی که الگوریتم از یک نقطه دلخواه بر روی تابعی که به دنبال مینیمم آن هستیم در جهت عکس گرادیان با ضریب مشخصی (که نرخ یادگیری – learning rate – نامیده میشود) شروع به حرکت میکند و طی چندین مرحله (در صورت همگرایی) به نقطه مینیمم تابع میرسد.



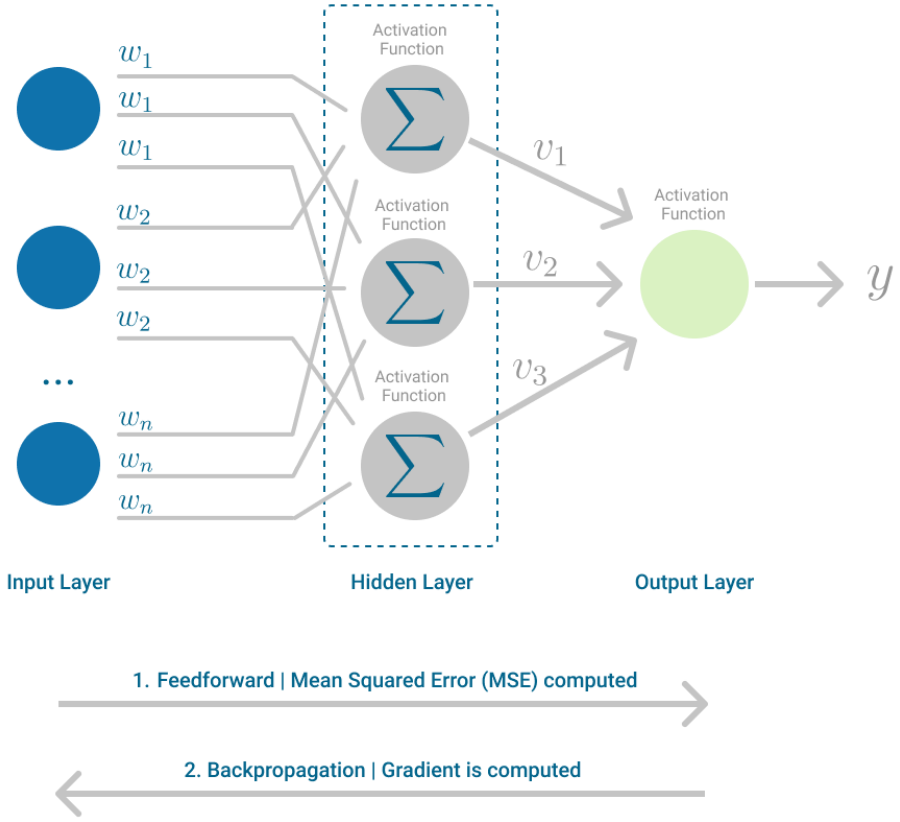
مسیر حرکت از نقطه شروع رندم تا نقطه مینیمم تابع از طریق رابطه زیر مشخص میشود:

$$w_{n+1} = w_n - \eta \nabla cost(w_n)$$

پس روش آموزش شبکه بدین صورت خلاصه میشود: در هر تکرار (که آن را epoch می نامیم) خروجی های تابع به ازای تمام ورودی ها را حساب کرده و مقدار تابع هزینه را حساب میکنیم. سپس از طریق رابطه gradient

descent در جهت کاهش هزینه حرکت میکنیم. این روند را آنقدر تکرار میکنیم تا تغییرات تابع هزینه به حداقل برسد و عملا به همگرایی برسیم.

اما سوال بعدی اینجاست که گرادیان تابع هزینه پیچیده یک شبکه عصبی را نسبت به ورودی های شبکه که پس از عبور از چندین لایه و چندین تابع فعالسازی به خروجی رسیده اند چطور حساب کنیم؟ این کار از طریق مکانیزم backpropagation انجام میشود. به بیان ساده برای محاسبه گرادیان خروجی نسبت به ورودی شبکه، از لایه آخر شروع کرده و لایه به لایه به عقب برمیگردیم. ارتباط بین گرادیان دو لایه متوالی از طریق مشتقات جزئی بدست می آید.



توضیحات بیشتر در این رابطه از حوصله آموزش این دستورکار خارج است. برای مطالعه بیشتر در این زمینه میتوانید به این لینک [1]مراجعه کنید.

[1] https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

در عمل با توجه به زیاد بودن داده های آموزشی، استفاده از تمام آن ها در هر مرحله مینیمم کردن تابع هزینه بسیار زمانبر است. به همین دلیل در هر مرحله آپدیت وزن ها (در هر epoch)، داده های آموزشی به چندین دسته کوچکتر (که آن ها را batch می نامیم) تقسیم میشوند و مینیمم کردن تابع هزینه در چندین مرحله (به تعداد batch های مورد استفاده) رخ میدهد. الگوریتم مورد استفاده در batch، mini-batch gradient descent نامیده میشود. (این عملیات با gradient descent که پیشتر توضیح دادیم کاملا یکسان است و تنها تفاوت تعداد داده ها میباشد.)

در بخش اول میخواهیم طبقه بندی احساسات را با استفاده از multilayer perceptron انجام دهیم. دیتاست مورد استفاده، دیتاست FER 2013 میباشد که شامل ۷ کلاس داده در ۳ گروه test،train و validation میباشد. دیتاست را میتوانید از این لینک [۲]دریافت کنید. در این آزمایش از کلاس disgust استفاده نمیکنیم.

برای کار با شبکه های عصبی در پایتون کتابخانه های متفاوتی وجود دارد، برای راحتی در این آزمایش از کتابخانه keras استفاده میکنیم. (استفاده از کتابخانه های دیگر مانند pytorch در صورت آشنایی دانشجو بلامانع است.) چنانچه با کتابخانه keras آشنایی ندارید برای شروع میتوانید به این لینک ها [۳،۴]مراجعه کنید.

ابتدا از هر کلاس دیتاست یک تصویر را رسم کنید. حال یک شبکه عصبی تک لایه (بدون لایه مخفی) برای طبقه بندی طراحی کنید. در هر epoch ، accuracy و loss بر روی داده های train و validation را گزارش کنید. انتخاب نوع optimizer، مقدار learning rate، تعداد epoch ها و اندازه هر batch به عهده شما است. در نهایت نمودار accuracy و loss در هر epoch را رسم کنید و روند همگرایی را مشاهده نمایید. برای رسم نمودار میتوانید از تابع زیر استفاده کنید:

[۲] https://drive.google.com/file/d/1NHSeb2w5D_9fxQQRKrmfGYnH0XU9uqkG/view?usp=sharing
[۳] https://machinelearningmastery.com/tutorial-first-neural-network-python-keras /
[۴] https://pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-keras /

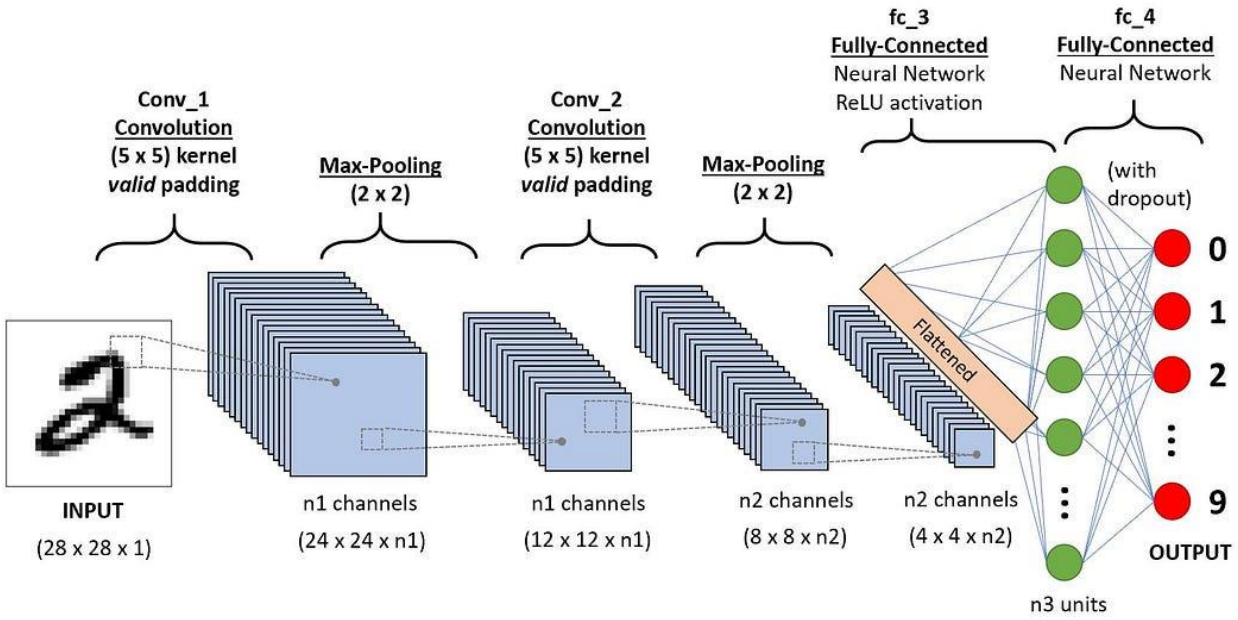
```python
def visualize_loss_and_acc(history):
  history_dict = history.history
  loss_values = history_dict['loss']
  val_loss_values = history_dict['val_loss']
  acc = history_dict['acc']

  epochs = range(1, len(acc) + 1)

  f = plt.figure(figsize=(10,3))

  plt.subplot(1,2,1)
  plt.plot(epochs, loss_values, 'bo', label='Training loss')
  plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
  plt.title('Training and validation loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()


  acc_values = history_dict['acc']
  val_acc = history_dict['val_acc']

  plt.subplot(1,2,2)
  plt.plot(epochs, acc, 'bo', label='Training acc')
  plt.plot(epochs, val_acc, 'b', label='Validation acc')
  plt.title('Training and validation accuracy')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()

  plt.show()
```

دقت مدل خود بر روی داده های test را اندازه گیری کنید و confusion matrix را نمایش دهید.

با اضافه کردن Hidden Layer به شبکه قسمت قبل، آیا می توانید خطای مرحله قبل را کم تر کنید؟ لایه (یا لایه های) مناسب را به شبکه خود اضافه کرده و نتایج قسمت قبل را مجددا گزارش کنید. (حداکثر ۴ لایه اضافه کنید.)

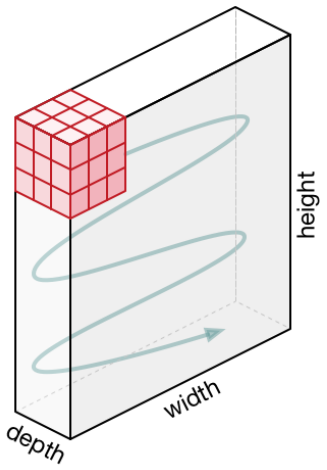
اثر تغییر optimizer، learning rate، اندازه batch و تعداد epoch ها را بررسی و با استناد به نتایج توضیح دهید.

در قسمت قبل دیدید که در آموزش MLP برای طبقه بندی تصاویر، ابتدا تصویر را تک بعدی کردیم و سپس به عنوان ورودی شبکه استفاده کردیم. این روش در رابطه با طبقه بندی های ساده و کلاس های نه چندان پیچیده پاسخگو است اما در حالت کلی روش ایده آلی برای یادگیری تصاویر و ویژگی های آن نیست. شبکه های کانولوشنی (Convolutional Neural Networks – CNNs) با حفظ شکل دو بعدی تصویر در ورودی شبکه و استفاده از فیلترها برای استخراج ویژگی های تصویر، وابستگی های بین پیکسل ها در تصویر را حفظ میکنند و گزینه مناسبتری برای طبقه بندی تصاویر هستند. نمای کلی از یک شبکه CNN ساده را در شکل زیر مشاهده میکنید:

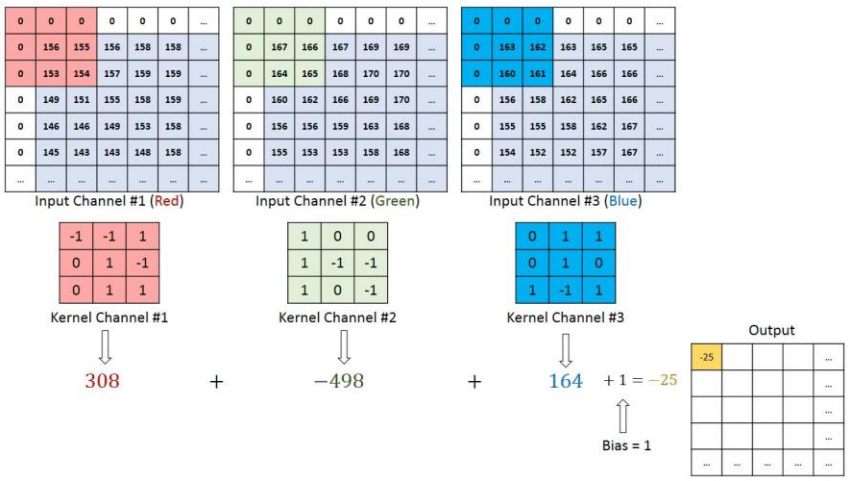


هر شبکه کانولوشنی از سه نوع لایه اساسی تشکیل شده است:

۱. Convolutional layer: در آزمایش های قبلی با فیلتر ها و نحوه اعمال آن ها به تصاویر آشنا شدید. یک لایه کانولوشنی از مجموعه ای از فیلترها تشکیل میشود که به ورودی لایه اعمال شده و خروجی را تشکیل میدهند.



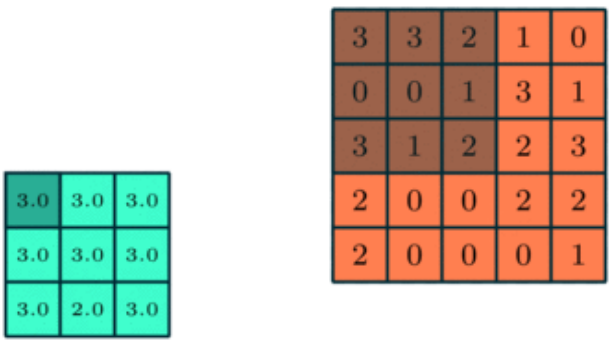
هر فیلتر میتواند در دو بعد ابعاد دلخواهی (۳*۳، ۵*۵، ۷*۷ و ...) داشته باشد. بعد سوم فیلتر را عمق لایه قبلی تعیین میکند. برای مثال اگر ورودی شبکه تصویر RGB با عمق ۳ باشد، فیلترهای تعریف شده (که kernel نامیده میشوند.) حتما عمق ۳ دارند. در تصویر زیر نمونه اعمال فیلتر با عمق ۳ به یک تصویر RGB را ملاحظه میکنید. هر عمق فیلتر در یک نقطه دلخواه به عمق متناظر خود در تصویر اعمال میشود. خروجی لایه کانولوشنی در آن نقطه از جمع حاصل خروجی سه فیلتر به علاوه یک مقدار ثابت (که bias نامیده میشود) بدست می آید.



واضح است که یک لایه کانولوشنی نسبت به یک لایه تماما متصل (fully connected – FC) در یک MLP وزن های کمتری برای یادگیری دارد و این امر فرآیند آموزش شبکه را راحتتر میکند.

خروجی هر لایه کانولوشنی مجموعه ای از feature map ها نامیده میشود که هر کدام حاصل اعمال یک فیلتر به ورودی لایه هستند.

۲. Pooling layer: وظیفه کاهش بعد feature map ها با حفظ حداکثر ویژگی های آن ها را دارد. در این لایه فیلتر جدیدی تعریف نمیشود (وزن های قابل یادگیری وجود ندارند!) اما کرنلی با ابعاد مشخص (مثلا ۳*۳) بر روی تصویر میلغزد و در پنجره ۳*۳ ماکزیمم گیری (Max Pooling) یا میانگین گیری (Average pooling) رخ میدهد.



در تصویر بالا کرنل ۳*۳ را مشاهده میکنید که از گوشه بالا چپ شروع به حرکت میکند و در هر پنجره ۳*۳ عمل ماکزیمم گیری را انجام میدهد.

۳. Fully Connected layer: تا به اینجا با حفظ وابستگی های دو بعدی بین پیکسل ها در ورودی و با حداقل وزن های قابل یادگیری ابعاد تصویر وردی را به حداقل رسانیدیم و ویژگی های تصویر را استخراج کردیم. حال در انتهای شبکه یک یا چند لایه تماما متصل برای امر طبقه بندی آموزش میدهیم.

برای مطالعه بیشتر درباره CNN ها میتوانید به این لینک[۵] مراجعه کنید.

[۵] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

برای آشنایی با نحوه پیاده سازی CNN ها در Keras میتوانید از این لینک [۶] کمک بگیرید.

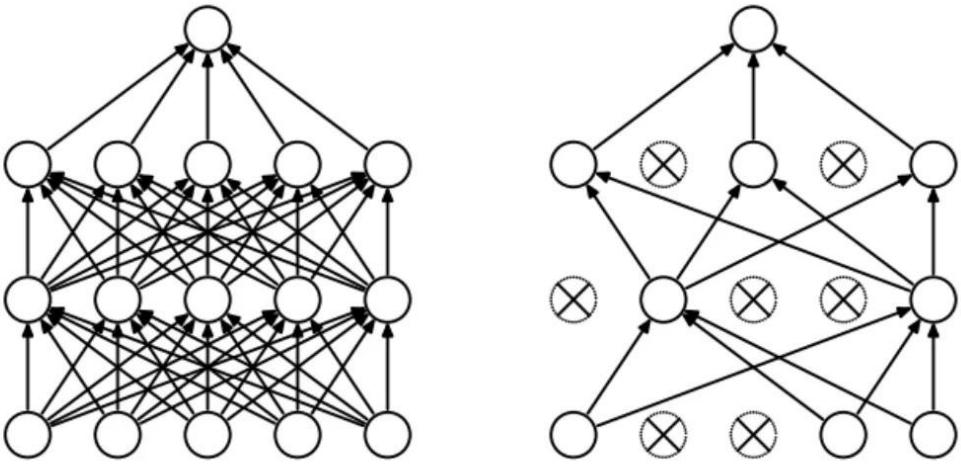
یک شبکه CNN دلخواه برای طبقه بندی احساسات طراحی کنید. شبکه مدنظر از حداکثر ۴ لایه کانولوشنی، تعدادی لایه pooling و حداکثر ۳ لایه FC تشکیل شده است. انتخاب سایز batch ، تعداد epoch ها، نوع optimizer و میزان learning rate به عهده شماست. نمودار loss و accuracy بهترین مدل خود (با بیشترین دقت یا کمترین loss بر روی داده های validation) را رسم کنید. این مدل را بر روی داده های تست نیز ارزیابی کنید و confusion matrix را نمایش دهید. (در این بخش شما باید حداقل ۴ حالت مختلف را بررسی و نتایج را گزارش کنید و در نهایت از بین حالات بررسی شده بهترین مدل را انتخاب کنید.)

(امتیازی) تعداد پارامترهای قابل یادگیری بهترین مدل MLP بخش قبل و بهترین CNN بخش بالا را محاسبه و تعداد آن ها را مقایسه کنید.

یکی از مشکلاتی که با عمیق کردن و یا به طور کلی افزایش تعداد پارامترهای قابل یادگیری شبکه رخ میدهد (مثلا با افزایش تعداد کرنل های هر لایه)، پیچیدگی بیش از اندازه مدل و احتمال رخ دادن overfitting است. بدین معنا که با افزایش تعداد epoch ها دقت شبکه بر روی داده های train افزایش می یابد اما نتایج بر روی داده های validation تغییری نمیکند. به بیانی دیگر شبکه به جای یادگیری ویژگی های کلی ورودی ها برای تصمیم گیری به حفظ کردن ویژگی های داده های آموزش روی آورده و نتیجتا قابلیت تعمیم دادن به داده های تست را از دست میدهد. برای رفع این مشکل در شبکه های عمیق، همانند سایر الگوریتم های یادگیری ماشین، از تکنیک های regularization استفاده میشود. این تکنیک ها با دادن پنالتی به رشد بی اندازه وزن های شبکه، سعی در حداقل کردن تابع هزینه و کاهش پیچیدگی مدل دارند. یکی از این روش ها استفاده از لایه dropout میباشد. به زبان ساده اضافه کردن لایه dropout در هنگام آموزش شبکه موجب خاموش کردن بعضی نورون ها در هر لایه با احتمالی مشخص میشود. مثلا در هر epoch در هر لایه، هر نورون با احتمال ۰.۵ ممکن است حذف شود پس شبکه در هر epoch با دسته ای متفاوت از نورون ها آموزش میبینند. این کار باعث میشود تا

[۶] https://learnopencv.com/implementing-cnn-tensorflow-keras/

همه نورون ها کم و بیش برای طبقه بندی هر نوع داده ای آموزش ببینند و وابستگی بین نورون ها برای جبران خطای یکدیگر در حالات خاص رخ داده در داده های آموزش کاهش یابد و نتیجتا قابلیت تعمیم به داده های تست افزایش پیدا کند.



(a) Standard Neural Net (b) After applying dropout.

به بهترین مدل CNN بخش قبل خود، لایه (یا لایه های) dropout اضافه کنید و تاثیر آن را در همگرایی نمودار loss و accuracy داده های train و validation و دقت روی داده های تست را توضیح دهید. (چنانچه مدل شما در قسمت قبل overfit نشده است میتوانید ابتدا با افزایش تعداد لایه ها یا افزایش تعداد فیلتر ها مدل را overfit کنید و سپس اثر افزایش لایه dropout را بهتر ببینید.)

(امتیازی) درباره سایر تکنیک های regularization تحقیق کنید و ۳ روش را به طور مختصر توضیح دهید.