# به نام خدا

گزارش تمرین سری پنجم درس بینایی کامپیوتر

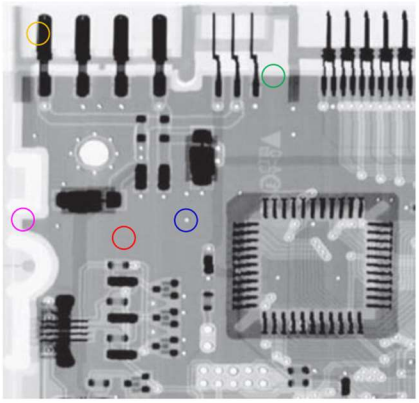
نام مدرس: دکتر محمدرضا محمدی

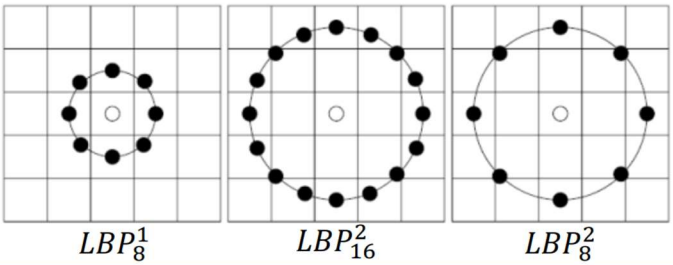
فرزان رحمانی ۹۹۵۲۱۲۷۱

سوال ۱

## الگوهای دودویی محلی

- یکی از متداول‌ترین ویژگی‌ها در حوزه تحلیل تصویر LBP است
- در این روش هر پیکسل توسط یک کد بازنمایی می‌شود
- کد LBP برای هر پیکسل از مقایسه مقدار آن پیکسل نسبت به مقدار پیکسل‌های همسایه بدست می‌آید





$LBP_8^1$ $LBP_{16}^2$ $LBP_8^2$

## الگوهای دودویی محلی

بزرگتر یا مساوی

- به هر پیکسل همسایه که کوچکتر از مقدار پیکسل مرکزی باشد عدد ۰ و به باقی پیکسل‌های همسایه عدد ۱ اختصاص می‌یابد
- کد نهایی، معادل با عدد دودویی است که از کنار هم قرار دادن این اعداد بدست می‌آید

| | | |
|---|---|---|
| 79 | 70 | 60 |
| 120 | 80 | 80 |
| 130 | 90 | 85 |

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | | 1 |
| 1 | 1 | 1 |

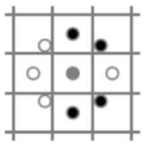
$$(00011111)_2 = 31$$

$$LBP_P^R(N_c) = \sum_{p=0}^{P-1} (N_p \geq N_c)\, 2^p$$

# LBP یکنواخت

- برخی از کدهای LBP مربوط به یک الگوی مشخص (مانند گوشه) هستند اما برخی الگوهای دیگر رفتار منظمی ندارند
- به الگوهایی بیش از ۲ تغییر بین صفر و یک داشته باشند غیریکنواخت گفته می‌شود
- در LBP هشت نقطه‌ای تعداد الگوهای یکنواخت ۵۸ عدد است و ۱۹۸ الگو غیریکنواخت وجود دارد
- در بسیاری از کاربردهای واقعی، بیش از ۹۰ درصد از الگوهای موجود در تصویر یکنواخت هستند
- بجای ۲۵۶ کد، از ۵۹ کد استفاده می‌شود (یک کد برای تمام الگوهای غیریکنواخت)

non-uniform



| | | |
|---|---|---|
| 1 | 0 | 1 |
| 1 | | 1 |
| 0 | 0 | 0 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | | 0 |
| 0 | 0 | 0 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | | 0 |
| 0 | 0 | 1 |

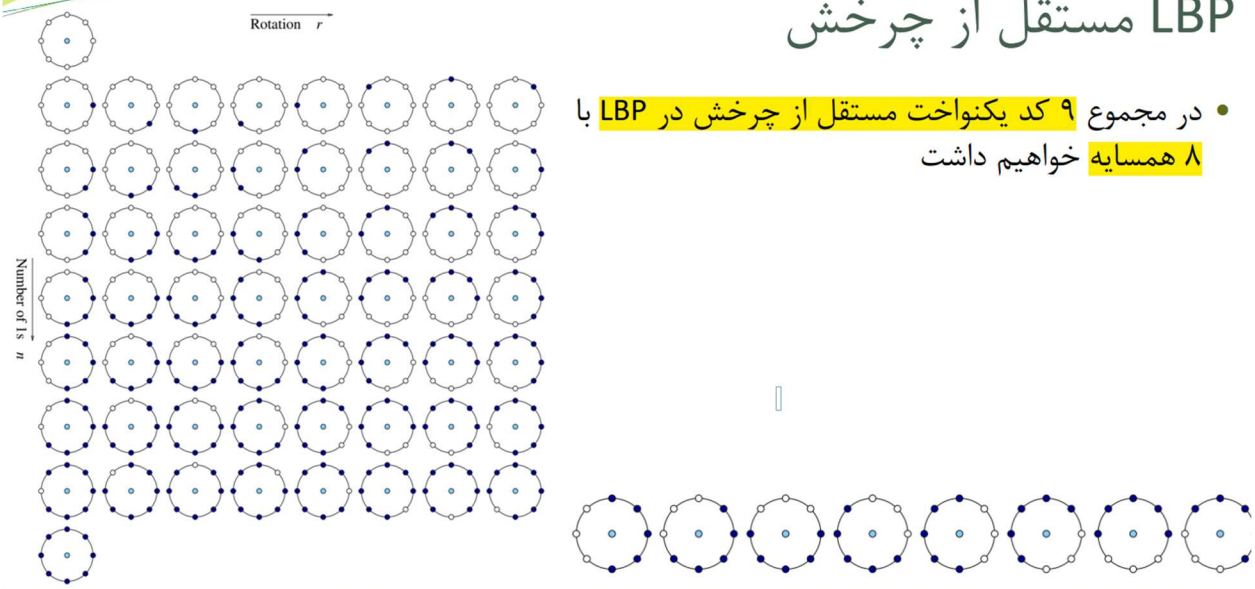
| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | | 1 |
| 0 | 1 | 1 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | | 1 |
| 1 | 1 | 1 |

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | | 0 |
| 1 | 0 | 1 |

# LBP مستقل از چرخش

- در مجموع ۹ کد یکنواخت مستقل از چرخش در LBP با ۸ همسایه خواهیم داشت

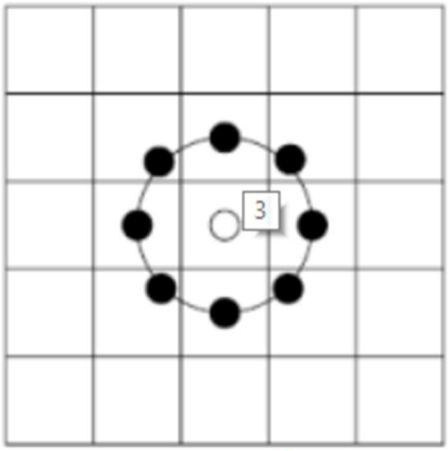



منبع:

https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_local_binary_pattern.html

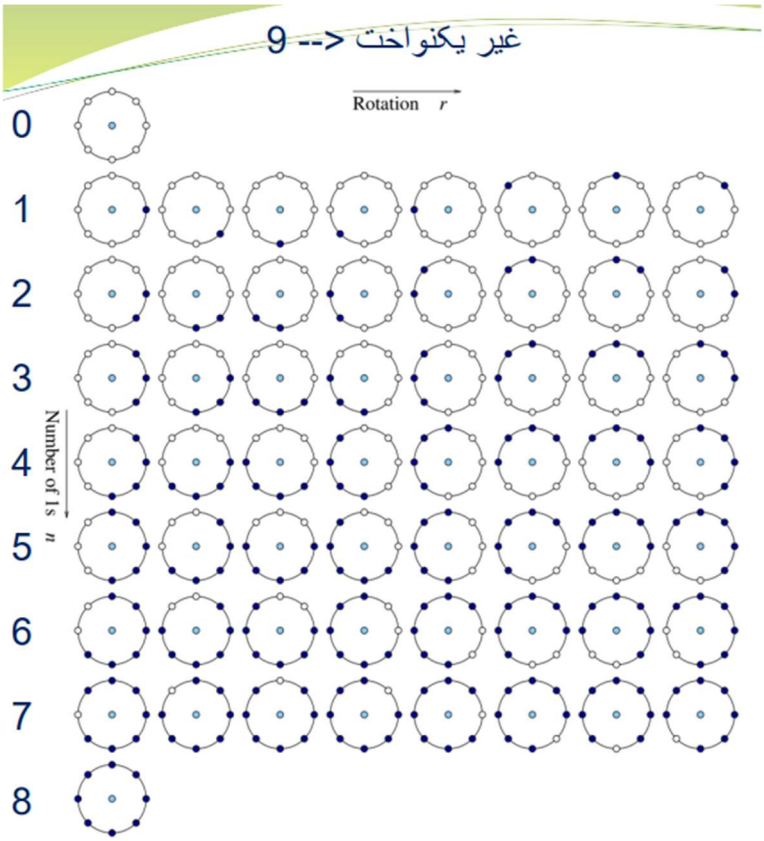
الف) بله نیاز به padding داریم. چرا که بعد از اعمال الگوریتم به هر پیکسل یک کد بازنمایی نگاشت می شود که اگر padding نزنیم پیکسل های مرزی و گوشه از بین می روند و بازنمایی نمی شوند. با توجه به اینکه شعاع ۱ است و کرنل سه در سه هست برای این تصویر نیاز است padding به طول یک پیکسل از بالا، پایین، چپ و راست بزنیم.(3 / 2 = 1)

نوع padding هم می تواند zero یا reflect یا ... باشد ولی ما در اینجا از zero padding استفاده می کنیم.

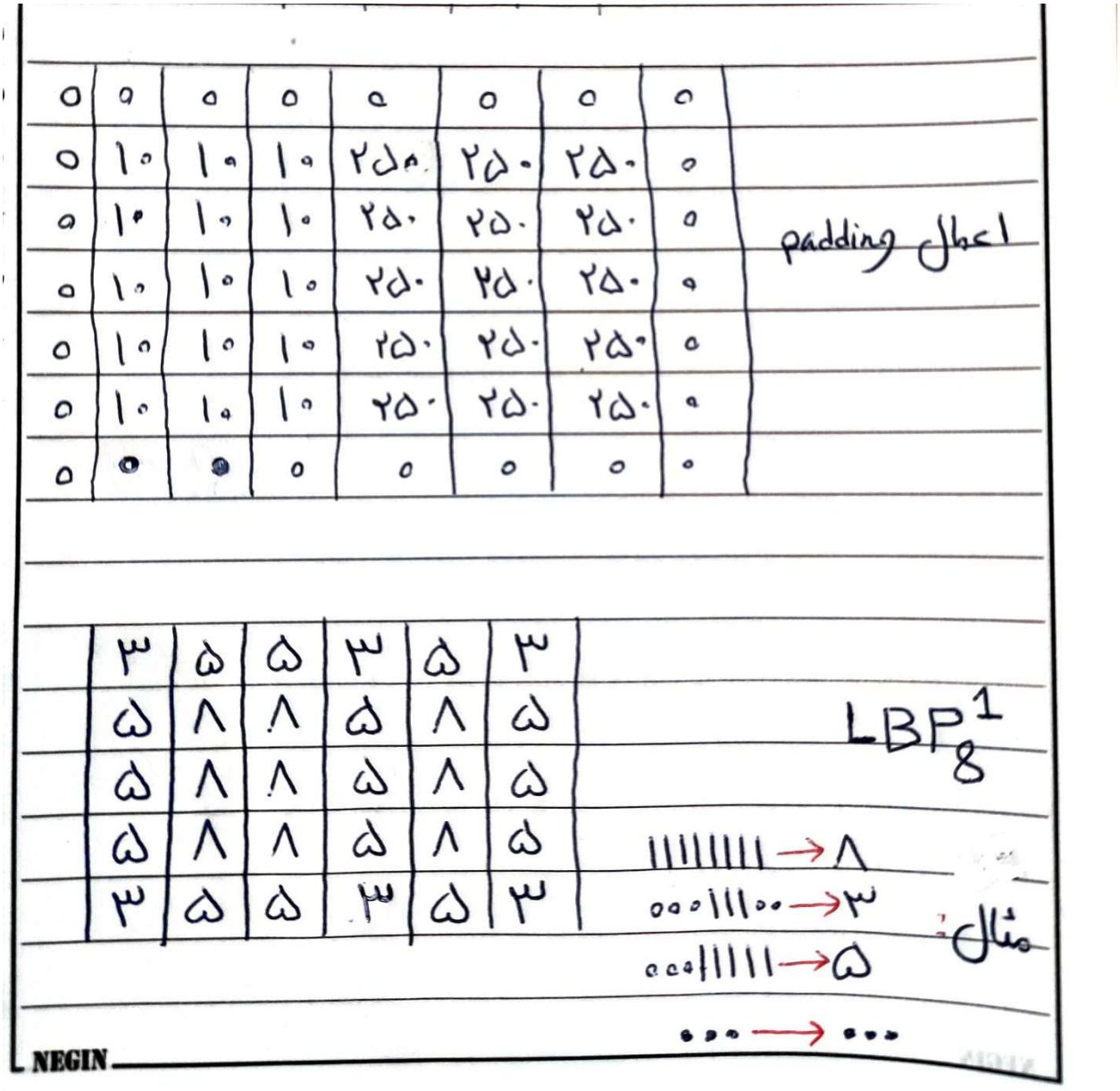



$LBP_8^1$

ب) فرض: از zero padding به طول یک واحد استفاده میکنیم.(3/2=1)
چون از حالت یکنواخت مستقل از چرخش که دارای هشت همسایه است استفاده میکنیم برای کد گذاری از روش زیر استفاده میکنیم که حالت غیر یکنواخت را با ۹ نشان می دهیم و بقیه حالات را به تعداد یک هایشان نگاشت میکنیم.



حل سوال:

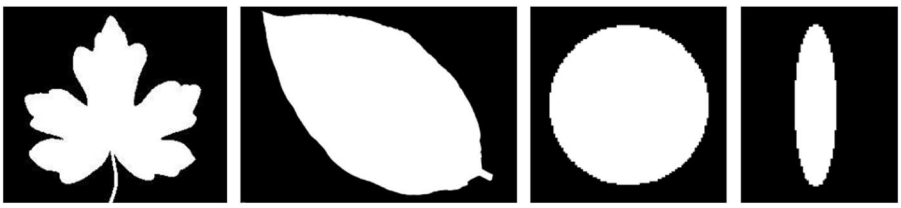

padding اعمال
LBP$_8^1$
مثال:
11111111 → ۸
00011100 → ۳
00011111 → ۵
000 → 000
NEGIN

سوال ۲

## فشردگی

- دایره یک شکل کاملا فشرده است و فشردگی یک شکل می‌تواند از مقایسه با آن بدست بیاید

$$Compactness = \frac{4\pi\ Area}{Perimeter^2}$$



0.21 0.60 1 0.52

## صلب بودن

- میزان چگال بودن یک شکل را ارزیابی می‌کند

به ازای محیط یکسان بیشترین مساحت را دارد

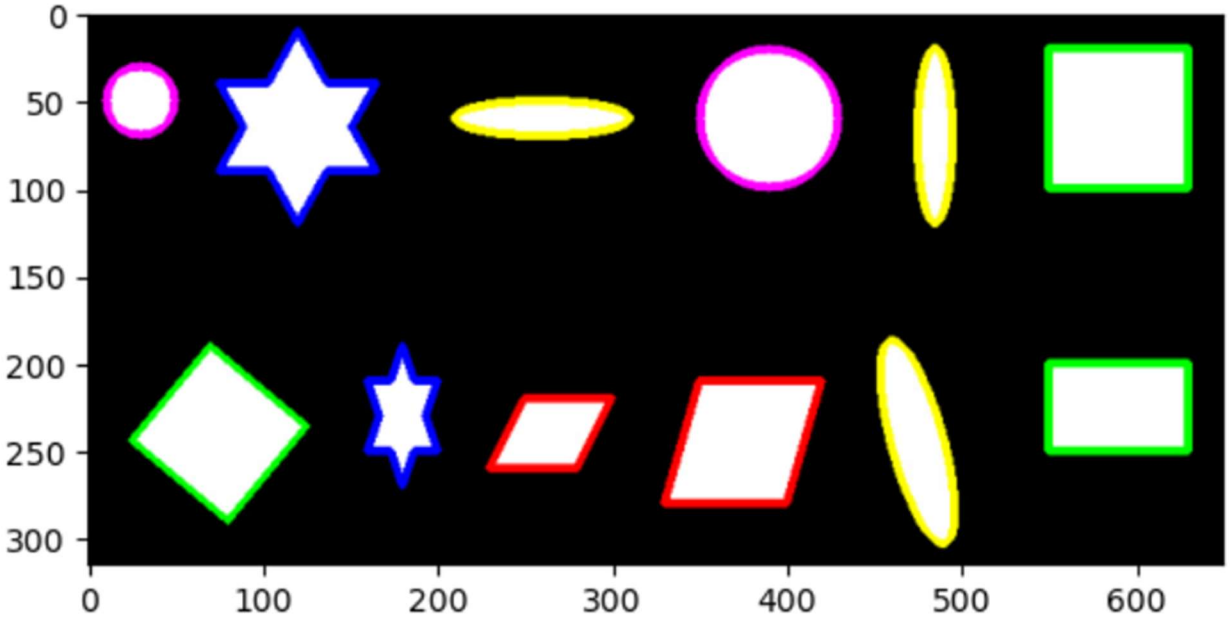
$$Solidity = \frac{Area}{ConvexArea}$$

0.72 0.97 1 1

ابتدا تصویر را میخوانیم و بعد از خاکستری کردن آستانه ۱۲۷ را اعمال می کنیم تا مقادیر صفر و یک شوند. بعد آن را نشان می دهیم که دارای ۱۲ شکل است. در قسمت بعد با استفاده از لینک داده شده و خواندن ویژگی های مختلفی را با استفاده از کانتور ورودی استخراج میکنیم که در مراحل بعدی به درد مان بخورد.( aspect_ratio, extent, solidity, equi_diameter, eccentricity, compactness, Area, Perimiter)

بعد از آن با توجه به توضیح داده شده در داخل نوت بوک به پیاده سازی تابع distance_criteria می پردازیم. اول با محاسبه MSE اختلاف بردار های ورودی پیاده سازی کردم ولی بعد از آن از تابع np.linalg.norm(x-y) استفاده کردم چرا که نتیجه بهتری میداد.

با تابع cv2.findContours(img,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE) کانتور های تصویر را درآوردیم و با چاپ ابعاد آن بهتر درکش کردیم.
در تابع :def extract_features_from_shapes(contours) ماتریس خروجی ۱۲*۲ را ساخته ایم.(با توجه به کامنت های نوت بوک و راهنمایی های آن) این ماتریس دارای ۱۲ آرایه از ویژگی های solidity, compactness است. از جایگشت های مختلفی از ویژگی ها برای محاسبه خروجی استفاده کردیم. ولی در نهایت با آزمایش و کمک گرفتن از تی ای های مربوطه به این نتیجه رسیدیم که این دو ویژگی بهتر عمل میکنند. سپس از این تابع و کانتور ها برای استخراج ویژگی ها استفاده کردم. تابع grouping توسط طراح عزیز پیاده سازی شده است. هم چنین تابع painting. در نهایت با بازی کردن با آستانه و آزمون و خطا به نتیجه مطلوب رسیدیم.



سوال ۳

منابع:

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

https://himanshuxd.medium.com/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e

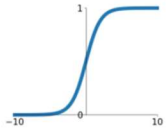
https://chat.openai.com/

## توابع فعال‌سازی

- به دلیل خطی بودن ضرب داخلی، وجود توابع فعال‌سازی غیرخطی ضروری است

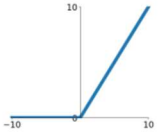
**Sigmoid**

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



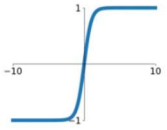
**ReLU**

$\max(0, x)$



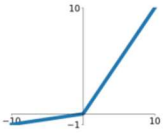
**tanh**

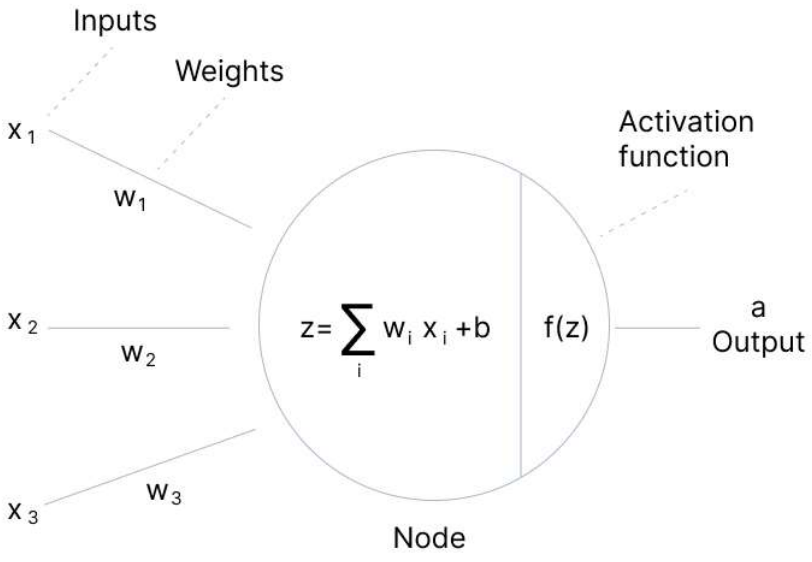
$\tanh(x)$



**Leaky ReLU**

$\max(0.1x, x)$






V7 Labs

توابع فعال سازی جزء حیاتی شبکه های عصبی هستند که غیرخطی بودن را وارد فرآیند تصمیم گیری شبکه می کنند. آنها به مجموع وزنی ورودی ها در هر نورون اعمال می شوند تا مشخص شود که آیا نورون باید فعال شود یا خیر. توابع فعال سازی به شبکه های عصبی کمک می کند تا الگوهای پیچیده را بیاموزند و آنها را گویاتر(more expressive) کنند. توابع فعال سازی سه هدف اصلی را دنبال می کنند:

۱. معرفی غیر خطی بودن(Introduce non-linearity): توابع فعال سازی شبکه های عصبی را قادر می سازد تا با معرفی غیرخطی، روابط پیچیده در داده ها را مدل کنند و یاد بگیرند. بدون توابع فعال‌سازی، شبکه فقط می‌تواند روابط خطی را یاد بگیرد و قابلیت‌های آن را به شدت محدود کند.
۲. فعال کردن بهینه‌سازی مبتنی بر گرادیان(Enable gradient-based optimization): توابع فعال‌سازی نقش مهمی در انتشار پس‌انداز دارند، که فرآیند به‌روزرسانی وزن‌های شبکه برای به حداقل رساندن خطا است. آنها مشتقات لازم برای محاسبه گرادیان ها را ارائه می دهند که سپس برای به روز رسانی وزنه ها در طول تمرین استفاده می شود.
۳. عادی سازی خروجی ها(Normalize outputs): توابع فعال سازی می توانند خروجی یک نورون را به یک محدوده خاص عادی کنند، که می تواند در سناریوهای خاصی مفید باشد. به عنوان مثال، فعال سازی sigmoid می تواند خروجی را بین ۰ و ۱ ترسیم کند، که برای مسائل طبقه بندی باینری که در آن خروجی احتمالات را نشان می دهد مفید است.

حالا بیایید در مورد توابع فعالسازی که ذکر کردید بحث کنیم:

۱. Sigmoid: تابع فعال سازی سیگموئید که به عنوان تابع لجستیک نیز شناخته می شود، ورودی را به مقداری بین ۰ و ۱ ترسیم می کند. فرمول ریاضی (f(x) = 1 / (1 + e^(-x)) دارد. خروجی تابع سیگموئید را می توان به عنوان یک احتمال تفسیر کرد. با این حال، توابع سیگموئید از مشکل "شیبی ناپدید شدن"("vanishing gradient") رنج می برند، جایی که گرادیان ها برای ورودی های با بزرگی های بزرگ بسیار کوچک می شوند، و یادگیری موثر برای شبکه های عمیق چالش برانگیز است.
   - تابع sigmoid یک ورودی با مقدار واقعی می گیرد و آن را در محدوده ای بین ۰ و ۱ له می کند.
   - دارای خروجی صاف و پیوسته است که آن را برای کاربردهایی که به خروجی احتمالی نیاز داریم مناسب می کند.
   - با این حال، از مشکل گرادیان ناپدید شدن(vanishing gradiant) رنج می برد، به این معنی که گرادیان ها برای مقادیر ورودی شدید بسیار کوچک می شوند، و یادگیری آن را برای شبکه سخت تر می کند.
۲. Tanh (Hyperbolic tangent): تابع فعال سازی tanh ورودی را به مقداری بین -۱ و ۱ ترسیم می کند. فرمول ریاضی (f(x) = (e^x - e^(-x)) / (e^x + e^(-x)) است. Tanh شبیه تابع سیگموئید است اما در مرکز ۰ است و محدوده خروجی متعادل تری را ارائه می دهد. مانند سیگموئید، tanh نیز از مشکل گرادیان ناپدید شدن(vanishing gradiant) رنج می برد.

- تابع tanh شبیه تابع سیگموئید است اما ورودی را بین -۱ و ۱ نگاشت می کند.
- مرکز صفر است که همگرا شدن الگوریتم های بهینه سازی را در مقایسه با تابع سیگموید آسان تر می کند.
- مانند تابع سیگموئید، از مشکل گرادیان ناپدید شدن نیز رنج می برد.

۳. ReLU (Rectified Linear Unit): واحد خطی اصلاح شده (ReLU) به صورت f(x) = max(0, x) تعریف می شود. تمام مقادیر منفی را روی ۰ تنظیم می کند و مقادیر مثبت را بدون تغییر می گذارد. ReLU به دلیل سادگی و کارایی محاسباتی محبوبیت پیدا کرده است. از مشکل شیب ناپدید شدن(vanishing gradient problem) رنج نمی برد و امکان آموزش سریعتر را فراهم می کند. با این حال، نورون های مبتنی بر ReLU می توانند در حین تمرین مستعد "مرگ" باشند، اگر در ۰ گیر کنند و هرگز بهبود نیابند، و منجر به سلول های عصبی مرده شوند که به یادگیری کمکی نمی کنند.

- تابع ReLU برای مقادیر ورودی منفی ۰ و برای مقادیر مثبت خود مقدار ورودی را برمی گرداند.
- از نظر محاسباتی کارآمد است و به کاهش مشکل گرادیان ناپدید کننده کمک می کند.
- ReLU به طور گسترده مورد استفاده قرار گرفته است و در بسیاری از کاربردها موفق بوده است.
- با این حال، ReLU می تواند از مشکل "ReLU در حال مرگ" ("dying ReLU") رنج ببرد، که در آن نورون ها می توانند برای همیشه در طول تمرین غیرفعال شوند و هرگز بهبود نخواهند یافت.

۴. PReLU (Parametric ReLU): واحد خطی اصلاح شده پارامتری (PReLU) توسعه ای از ReLU است که یک پارامتر قابل یادگیری را به تابع معرفی می کند. به جای تنظیم مقادیر منفی روی ۰، PReLU از یک تابع خطی برای ورودی های منفی استفاده می کند و به برخی از مقادیر منفی اجازه عبور می دهد. فرمول ریاضی f(x) = max(0, x) + a * min(0, x)، که در آن 'a' یک پارامتر قابل یادگیری است. PReLU با اجازه دادن به برخی از گرادیان های منفی در طول تمرین به کاهش مشکل در حال مرگ ReLU کمک می کند.

- PReLU تعمیمی از ReLU است که در آن قسمت منفی تابع پارامتر شده است.
- شیب کوچکی را برای ورودی های منفی معرفی می کند، که اجازه می دهد مقداری گرادیان حتی برای مقادیر منفی جریان یابد.
- PReLU به حل مشکل ReLU در حال مرگ(dying ReLU) کمک می کند و در موارد خاص عملکرد بهتری را نشان داده است.

مقایسه: توابع سیگموئید و tanh در سناریوهایی که نیاز دارید خروجی‌ها در محدوده خاصی باشند، مانند وظایف طبقه‌بندی باینری، مفید هستند. با این حال، به دلیل مشکل گرادیان ناپدید شدن، کمتر در لایه‌های پنهان شبکه‌های عصبی عمیق استفاده می‌شوند.

ReLU و انواع آن، از جمله PReLU، به طور گسترده در شبکه‌های عصبی عمیق استفاده می‌شوند، زیرا به مشکل گرادیان ناپدید شدن رسیدگی می‌کنند و امکان آموزش کارآمد را فراهم می‌کنند. ReLU ساده

ترین و متداول ترین تابع فعال سازی است، در حالی که PReLU با یادگیری پارامتر شیب منفی درجه کمی از انعطاف پذیری را معرفی می کند.

مقایسه عملکردهای فعال سازی:

- Sigmoid و tanh توابع محدودی هستند و معمولاً از مشکل گرادیان ناپدید شدن رنج می برند، به خصوص برای شبکه های عصبی عمیق.
- ReLU و PReLU از مشکل گرادیان ناپدید شدن رنج نمی برند و به انتخاب های محبوب در یادگیری عمیق تبدیل شده اند.
- ReLU از نظر محاسباتی کارآمد است اما می تواند منجر به سلول های عصبی مرده شود. PReLU با معرفی یک شیب منفی کوچک به این موضوع می پردازد.
- ReLU و PReLU به دلیل سادگی و عملکرد آموزشی بهتر معمولاً برای اکثر برنامه ها ترجیح داده می شوند.
- Sigmoid و tanh هنوز در موارد خاص مفید هستند، مانند طبقه بندی باینری یا زمانی که یک محدوده خروجی محدود مورد نظر است.

به طور کلی، انتخاب تابع فعال سازی به مسئله خاص(specific problem)، معماری شبکه( network architecture) و ویژگی های داده های مدل سازی شده( properties of the data being modeled) و ویژگی های مجموعه داده(dataset) مورد استفاده بستگی دارد. آزمایش، تنظیم و تحلیل تجربی معمولاً برای تعیین مناسب‌ترین تابع فعال‌سازی برای یک کار معین مورد نیاز است.

به زبان انگلیسی(اصطلاحات قابل فهم ترند):

Activation functions are a vital component of neural networks that introduce non-linearity into the network's decision-making process. They are applied to the weighted sum of inputs at each neuron to determine whether the neuron should be activated or not. Activation functions help neural networks learn complex patterns and make them more expressive. Activation functions serve three main purposes:

1. Introduce non-linearity: Activation functions enable neural networks to model and learn complex relationships in data by introducing non-linearities. Without activation functions, the network would only be able to learn linear relationships, severely limiting its capabilities.
2. Enable gradient-based optimization: Activation functions play a crucial role in backpropagation, which is the process of updating the network's weights

to minimize the error. They provide derivatives necessary for computing gradients, which are then used to update the weights during training.

3. Normalize outputs: Activation functions can normalize the output of a neuron to a specific range, which can be helpful in certain scenarios. For example, sigmoid activation can map the output between 0 and 1, which is useful for binary classification problems where the output represents probabilities.

Now, let's discuss the activation functions you mentioned:

1. Sigmoid: The sigmoid activation function, also known as the logistic function, maps the input to a value between 0 and 1. It has the mathematical formula f(x) = 1 / (1 + e^(-x)). The output of the sigmoid function can be interpreted as a probability. However, sigmoid functions suffer from the "vanishing gradient" problem, where gradients become extremely small for inputs with large magnitudes, making it challenging for deep networks to learn effectively.
    - The sigmoid function takes a real-valued input and squashes it into a range between 0 and 1.
    - It has a smooth and continuous output, which makes it suitable for applications where we need a probability-like output.
    - However, it suffers from the vanishing gradient problem, meaning that gradients become very small for extreme input values, making it harder for the network to learn.

2. Tanh (Hyperbolic tangent): The tanh activation function maps the input to a value between -1 and 1. It has the mathematical formula f(x) = (e^x - e^(-x)) / (e^x + e^(-x)). Tanh is similar to the sigmoid function but centered at 0, offering a more balanced output range. Like the sigmoid, tanh also suffers from the vanishing gradient problem.
    - The tanh function is similar to the sigmoid function but squashes the input between -1 and 1.

- It is zero-centered, which makes it easier for optimization algorithms to converge compared to the sigmoid function.
- Like the sigmoid function, it also suffers from the vanishing gradient problem.

3. ReLU (Rectified Linear Unit): The Rectified Linear Unit (ReLU) is defined as f(x) = max(0, x). It sets all negative values to 0 and leaves positive values unchanged. ReLU has gained popularity due to its simplicity and computational efficiency. It does not suffer from the vanishing gradient problem and allows for faster training. However, ReLU-based neurons can be susceptible to "dying" during training if they get stuck at 0 and never recover, leading to dead neurons that don't contribute to learning.
    - The ReLU function returns 0 for negative input values and the input value itself for positive values.
    - It is computationally efficient and helps alleviate the vanishing gradient problem.
    - ReLU has been widely adopted and has proven successful in many applications.
    - However, ReLU can suffer from the "dying ReLU" problem, where neurons can become permanently inactive during training and never recover.
4. PReLU (Parametric ReLU): The Parametric Rectified Linear Unit (PReLU) is an extension of ReLU that introduces a learnable parameter to the function. Instead of setting negative values to 0, PReLU uses a linear function for negative inputs, allowing some negative values to pass through. The mathematical formula is f(x) = max(0, x) + a * min(0, x), where 'a' is a learnable parameter. PReLU helps alleviate the dying ReLU problem by allowing some negative gradients to flow during training.
    - PReLU is an extension of ReLU where the negative part of the function is parameterized.

- It introduces a small slope for negative inputs, allowing some gradient to flow even for negative values.
- PReLU helps address the dying ReLU problem and has shown improved performance in certain cases.

In comparison, sigmoid and tanh functions are useful in scenarios where you need outputs to be within a specific range, such as binary classification tasks. However, they are less commonly used in hidden layers of deep neural networks due to the vanishing gradient problem.

ReLU and its variants, including PReLU, are widely used in deep neural networks because they address the vanishing gradient problem and allow for efficient training. ReLU is the simplest and most commonly used activation function, while PReLU introduces a small degree of flexibility by learning the negative slope parameter.

Comparing the activation functions:

- Sigmoid and tanh are bounded functions and tend to suffer from the vanishing gradient problem, especially for deep neural networks.
- ReLU and PReLU do not suffer from the vanishing gradient problem and have become popular choices in deep learning.
- ReLU is computationally efficient but can lead to dead neurons. PReLU addresses this issue by introducing a small negative slope.
- ReLU and PReLU are typically preferred for most applications due to their simplicity and better training performance.
- Sigmoid and tanh are still useful in specific cases, such as binary classification or when a bounded output range is desired.

Overall, the choice of activation function depends on the specific problem, network architecture, and the properties of the data being modeled, and the characteristics of the dataset being used. Experimentation, tuning and empirical analysis are typically required to determine the most suitable activation function for a given task.

## سوال ۴

کد داده شده دارای کامنت های کامل بود و براحتی با خواندن آن متوجه شدم چه کاری انجام می دهد.
آنالیز کد داده شده روی داده های fashion MNIST به طور اجمالی:

۱. ابتدا با سرچ کردن و خواندن لینک های داده شده مرتبط، مسئله را بهتر متوجه شدم.
۲. سپس کتابخانه های لازم و داده fashion_mnist را import کردیم.
۳. سپس داده ها را داخل RAM لود کردیم.
۴. کلاس های مربوطه را داخل یک ارایه گذاشتیم تا بعد از یافتن اندیس آن بفهمیم که مرتبط به کدام کلاس است و فهم بهتری داشته باشیم.
۵. ابعاد داده های تست و آموزش را دیدیم.
۶. چند تا از داده ها را همراه با برچسب شان نمایش دادیم که بهتر درک کنیم.
۷. داده ها را flat و سپس normalize کردیم.(preprocessing)
۸. با استفاده از keras.models مدلی sequential ساختیم.
۹. خلاصه مدل را چاپ کردیم تا پارامتر ها و ساختارش را ببینیم.
۱۰. مدل ها را با optimizer آدام و تابع ضرر کراس انتروپی compile کردیم. ورودی from_logits=True برای این است که در تعریف مدل از softmax استفاده نکردیم.
۱۱. مدل را در 10 epoch آموزش دادیم و fit کردیم.
۱۲. نمودار های Train losses و train accuracies را با استفاده از خروجی تابع model.fit رسم کردیم.
۱۳. حال دقت مدل را روی داده های تست می سنجیم با استفاده از model.evaluate و ارزیابی می کنیم.
۱۴. سپس بعد از افزودن لایه softmax از مدل برای پیش بینی کردن داده های تست استفاده کردیم.
۱۵. یکی از پیش بینی ها را دیدیم که آرایه ای ۱۰ عضوی است.
۱۶. اولین داده تست را همراه با پیش بینی نشان دیدیم و میبینیم که مدل درست عمل کرده است.

بعد از خواندن کد های داده شده و لینک داده شده در مورد functional models به طور مشابه با کد های داده شده به پیاده سازی پرداختم.( داخل کد کامنت های توضیح هم موجود است.)
پیاده سازی مدل های خواسته شده:

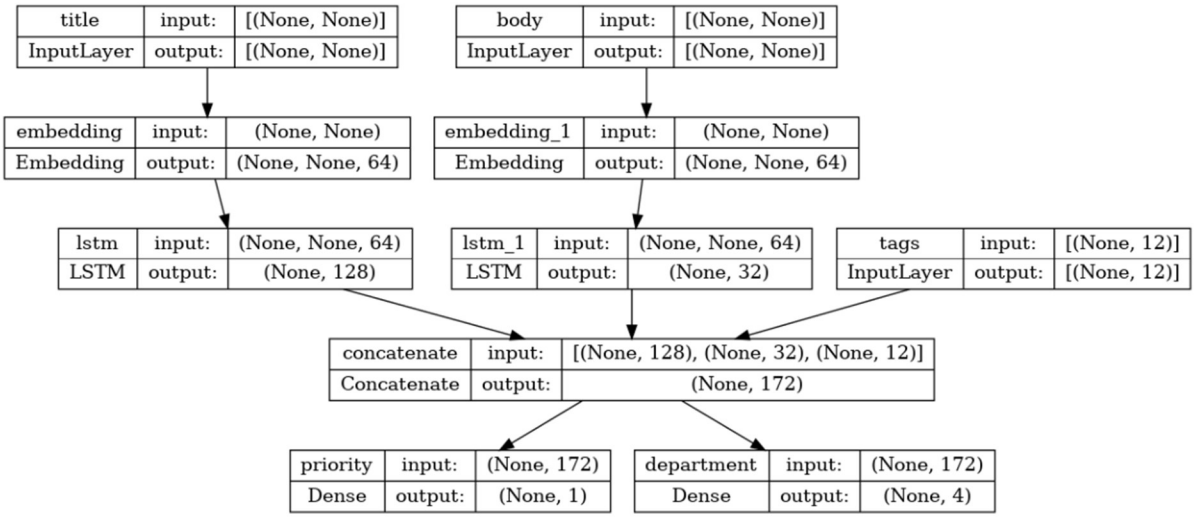
۱. ابتدا با سرچ کردن و خواندن لینک های داده شده مرتبط، مسئله را بهتر متوجه شدم.
۲. سپس کتابخانه های لازم و داده mnist را import کردیم.
۳. سپس داده ها را داخل RAM لود کردیم.
۴. داده اول train را همراه با برچسبش نمایش دادیم تا درک بهتری داشته باشیم.(با استفاده از pyplot)
۵. ابعاد داده های تست و آموزش را دیدیم.
۶. داده ها را flat(یک بعدی) و سپس normalize(تقسیم بر ۲۵۵ و تبدیل به float) کردیم.(preprocessing) (از reshape کمک گرفتیم)
۷. با استفاده از keras.models مدلی sequential ساختیم.
۸. لایه ورودی و دو لایه پنهان dense و لایه dense آخر را به عنوان معماری مدل تعریف کردیم.(از توابع فعال ساز relu, softmax استفاده کردیم.)
۹. خلاصه مدل را چاپ کردیم تا پارامتر ها و ساختارش را ببینیم.
۱۰. مدل ها را با rmsprop optimizer و تابع ضرر SparseCategoricalCrossentropy compile کردیم. ورودی from_logits=False برای این است که در تعریف مدل از softmax استفاده کردیم.
۱۱. با استفاده از keras.utils.plot_model(seq_model, "my_first_model_with_shape_info.png", show_shapes=True) ساختار مدل را کشیدیم.
۱۲. مدل را در 10 epoch آموزش دادیم و fit کردیم.
۱۳. نمودار های Train losses و train accuracies را با استفاده از خروجی تابع model.fit رسم کردیم.

۱۴. حال دقت مدل را روی داده های تست می سنجیم با استفاده از model.evaluate و ارزیابی می کنیم.(می بینیم که به دقت ۹۷ درصد رسیده است.)
۱۵. سپس از مدل برای پیش بینی کردن داده های تست استفاده کردیم.
۱۶. اولین داده تست را همراه با پیش بینی نشان دیدیم و میبینیم که مدل درست عمل کرده است.
۱۷. برای پیاده سازی functional ابتدا لایه ورودی را تعریف کردیم. سپس لایه dense بعدی را تعریف کردیم و به عنوان ورودی لایه input را دادیم که معادل این است که خروجی input به ورودی این تابع وصل شود. سپس لایه بعدی را به همین شیوه تعریف کردیم. در نهایت لایه آهر را تعریف کردیم. بعد از آن با استفاده از keras.Model(inputs=inputs, outputs=outputs, name="functional_model") مدل را تعریف کردیم همان طور که میبینیم برا تعریف فقط لایه های ورودی و خروجی را باید به ورودی پاس بدهیم.
۱۸. خلاصه مدل را چاپ کردیم تا پارامتر ها و ساختارش را ببینیم.
۱۹. با استفاده از keras.utils.plot_model(seq_model, "my_first_model_with_shape_info.png", show_shapes=True) ساختار مدل را کشیدیم.
۲۰. مدل ها را با rmsprop optimizer و تابع ضرر SparseCategoricalCrossentropy compile کردیم. ورودی from_logits=True برای این است که در تعریف مدل از softmax استفاده نکردیم.
۲۱. مدل را در 10 epoch آموزش دادیم و fit کردیم.
۲۲. نمودار های Train losses و train accuracies را با استفاده از خروجی تابع model.fit رسم کردیم.
۲۳. حال دقت مدل را روی داده های تست می سنجیم با استفاده از model.evaluate و ارزیابی می کنیم.(می بینیم که به دقت ۹۷ درصد رسیده است.)
۲۴. سپس از مدل برای پیش بینی کردن داده های تست استفاده کردیم.
۲۵. اولین داده تست را همراه با پیش بینی نشان دیدیم و میبینیم که مدل درست عمل کرده است.

## سوال ۵

خیر، نمی توانیم پیاده سازی کنیم. چرا که متد functional انعطاف بیشتری(more flexible) نسبت به متد sequential دارند. در واقع متد functional می‌تواند مدل‌هایی با توپولوژی غیرخطی(non-linear topology)، لایه‌های مشترک(shared layers) و حتی چندین ورودی یا خروجی( multiple inputs or outputs) را مدیریت کند. پس متد functional طیف وسیع تری از مدل ها را شامل می شود. لذا متد sequential زیر مجموعه متد functional است یعنی هر شبکه ای که بتوانیم به صورت sequential پیاده سازی کنیم، می توانیم به صورت functional نیز پیاده سازی کنیم ولی عکس این قضیه لزوما برقرار نیست. همچین بعضی از شبکه ها دارای skip connection هستند که با متد functional قابل پیاده سازی نیست.

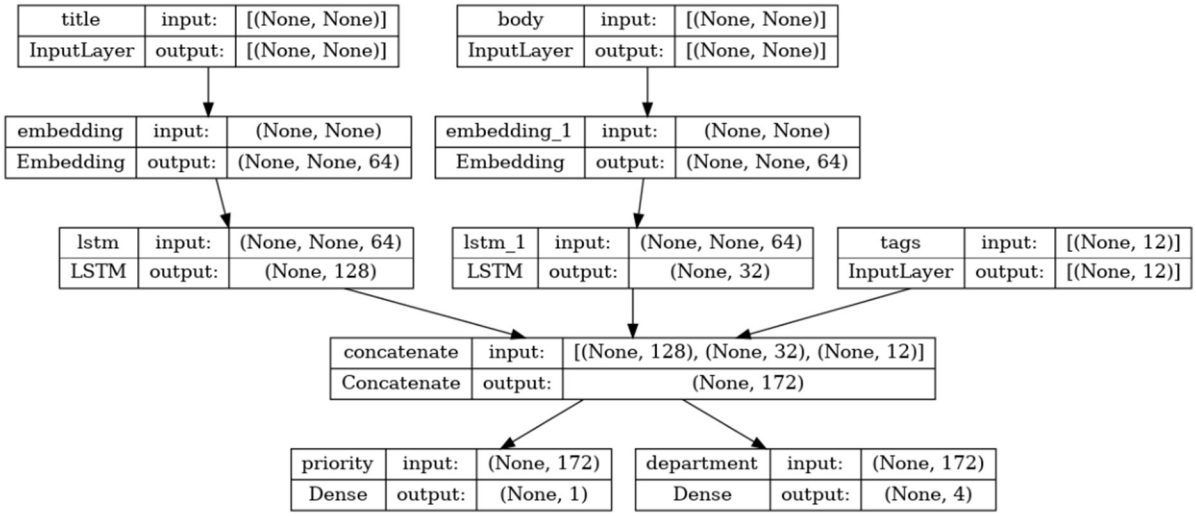
مثال نقض زیر برای پاسخ به این سوال قابل ارائه است:



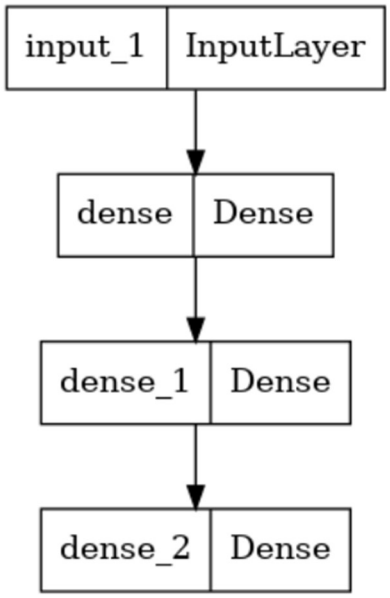

شبکه بالا با روش تابعی (functional method) پیاده سازی شده است و همان طور که می بینید دارای چندین ورودی و چندین خروجی است ، لایه مشترک(shared layer) دارد و لایه های میانی نیز فقط دارای یک لایه ورودی و خروجی نیستند. این شبکه به روش sequential قابل پیاده سازی نیست و مثال نقض سوال محسوب می شود.

تعریف متد های functional و sequential:

۱. روش تابعی :(functional method) در روش تابعی، شبکه عصبی با استفاده از کلاس‌هایی که در فریمورک‌های مختلف موجود هستند، به صورت تابعی تعریف می‌شود. در این روش، لایه‌ها و ارتباطات بین آنها به صورت جداگانه مشخص می‌شوند و توابع فعال‌سازی و سایر اجزا نیز به صورت جداگانه قابل تنظیم هستند. در نهایت، با اتصال لایه‌ها و ارتباطات بین آنها، یک شبکه عصبی کامل ساخته می‌شود.در واقع ایده اصلی این است که یک مدل یادگیری عمیق معمولاً یک گراف غیر چرخه ای جهت دار (DAG) از لایه ها است. بنابراین روش تابعی (functional method) راهی برای ساخت نمودارهای لایه ها است.




۲. روش ترتیبی :(sequential method) در روش ترتیبی، شبکه عصبی با استفاده از مدل‌های توالی‌ای مانند مدلهای ترتیبی در فریمورک‌های مختلف تعریف می‌شود. در این روش، لایه‌ها به ترتیب قرار می‌گیرند و خروجی یک لایه به عنوان ورودی لایه بعدی استفاده می‌شود. با اضافه کردن لایه به لایه، شبکه به طور ترتیبی ساخته می‌شود.مثال:




همچنین می شود در طراحی یک شبکه عصبی، روش‌های تابعی (functional) و ترتیبی (sequential)می‌توانند با هم ترکیب شوند. به عنوان مثال، می‌توان با استفاده از روش تابعی، لایه‌ها و ارتباطات آنها را به صورت جداگانه تعریف کرده و سپس با استفاده از روش ترتیبی، آنها را به ترتیب اضافه کرد تا یک مدل کامل ساخته شود.

با استفاده از ترکیب این دو روش، امکانات و قدرت طراحی شبکه عصبی افزایش می‌یابد. از روش تابعی می‌توان برای اعمال تنظیمات جزئی و سفارشی‌سازی دقیق‌تر استفاده کرد، در حالی که روش ترتیبی به وفور استفاده می‌شود برای ساختاردهی و مدیریت ساده‌تر شبکه در مراحل پیش‌پردازش، عملیات مشابه در لایه‌ها و غیره.

منابع:


https://towardsdatascience.com/a-comprehensive-introduction-to-tensorflows-sequential-api-and-model-for-deep-learning-c5e31aee49fa
https://machinelearningmastery.com/keras-functional-api-deep-learning
https://www.analyticsvidhya.com/blog/2021/07/understanding-sequential-vs-functional-api-in-keras/


سوال ۶

الف) چون تصویر دارای ۳ کانال هست فرض میکنیم که کرنل نیز ۷ در ۷ در ۳ است. با توجه به اینکه padding نمیزنیم ابعاد آن کاهش می یابد(کانولوشن فقط برای پیکسل مرکزی زده می شود) و خروجی یک عدد(یک مقدار فعال سازی واحد) می شود یعنی یک بعدی می شود.

7 x 7 x 3 → 1 x 1 x 1 (عدد)

If we convolve the 7x7x3 image with a 7x7x3 kernel using a stride of 1, the output will have dimensions of (7 - 7 + 1) x (7 - 7 + 1) x (3 - 3 + 1) = 1x1x1. The output will be a single activation value.

ب) چون تصویر دارای ۳ کانال هست فرض میکنیم که کرنل مرحله اول نیز ۳ در ۳ در ۳ است و کرنل دو مرحله بعد ۳ در ۳ هستند. با توجه به اینکه padding نمیزنیم ابعاد تصویر در هر مرحله کاهش می یابد. پس از سه مرحله خروجی یک عدد(یک مقدار فعال سازی واحد) می شود یعنی یک بعدی می شود.

7 x 7 x 3 → 5 x 5 x 1 → 3 x 3 x 1 → 1 x 1 x 1 (عدد)

If we convolve the 7x7x3 image with three 3x3 kernels in three steps, the dimensions of the output will depend on the stride used and the padding applied. Let's assume we use a stride of 1 and no padding. In first step, the output size will be (7 - 3 + 1) x (7 - 3 + 1) x (3 -3 + 1) = 5x5x1. In second step, the output size will be (5 - 3 + 1) x (5 - 3 + 1) x (3 -3 + 1) = 3x3x1. In third step, the output size will be (3 - 3 + 1) x (3 - 3 + 1) x (3 -3 + 1) = 1x1x1. So after performing three convolutions, the final output will have dimensions of 1x1x1(a single activation value).

ج)

مقایسه دو بخش اول از نظر تعداد پارامترها و کیفیت ویژگی های استخراج شده:

• تعداد پارامترها: در قسمت اول، هنگام کانوالو تصویر با یک کرنل ۷ در ۷، تعداد پارامترها با فرض سه کانال ورودی 147 = 3 x 7 x 7 خواهد بود. در بخش دوم، هنگام کانوالو تصویر با سه کرنل ۳ در ۳، با فرض سه کانال ورودی، تعداد پارامترها 45 = 1 x 3 x 3 + 1 x 3 x 3 + 3 x 3 x 3 خواهد بود. بنابراین، رویکرد دوم(سه کرنل ۳ در ۳) به پارامترهای کمتری نیاز دارد.(147 > 45)

• کیفیت ویژگی های استخراج شده: رویکرد دوم با سه کرنل ۳ در ۳ در سه مرحله به طور بالقوه می تواند ویژگی های متنوع و غیرخطی بیشتری را در مقایسه با رویکرد اول با یک کرنل ۷ در ۷ استخراج کند. هر کرنل ۳ در ۳ بر روی یک فیلد گیرنده(receptive field) خاص تمرکز می کند و فیلترهای مختلف را می آموزد که امکان نمایش ویژگی های غنی تر را فراهم می کند. مراحل چندگانه با فعال‌سازی‌های غیرخطی(multiple steps with non-linear activations)، تبدیل عمیق‌تر و غیرخطی‌تر(deeper and more non-linear) ورودی را فراهم می‌کند و شبکه را قادر می‌سازد تا الگوها و روابط پیچیده‌ای(complex patterns and relationships) را ثبت کند.

با توجه به تعداد پارامترها و کیفیت ویژگی های استخراج شده، رویکرد دوم با چندین کرنل ۳ در ۳ به طور کلی در طراحی یک شبکه موثرتر خواهد بود. نمایش فشرده‌تر(compact representation) و ظرفیت بیشتری برای ثبت ویژگی‌های متنوع و غیرخطی ارائه می‌دهد که به طور بالقوه منجر به بهبود عملکرد در کارهایی مانند طبقه‌بندی تصویر یا تشخیص اشیا می‌شود.

پایان