

به نام خدا

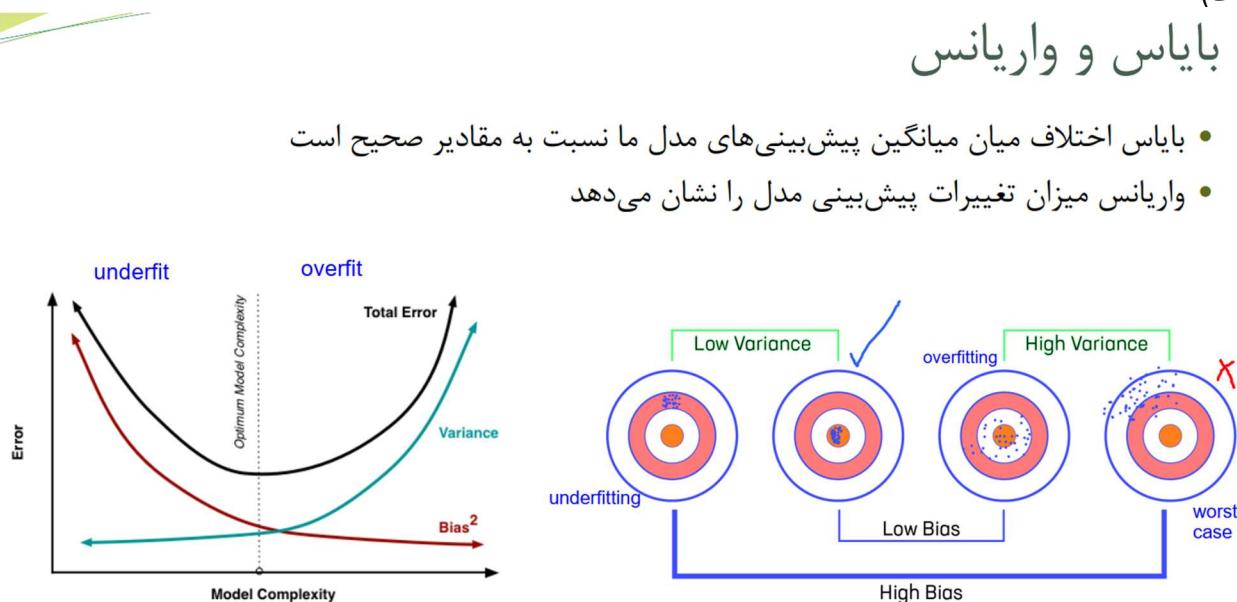
تمرین سری دوم
درس مبانی یادگیری عمیق
دکتر مرضیه داود آبادی

فرزان رحمانی
۹۹۵۲۱۲۷۱

سوال اول
(الف)

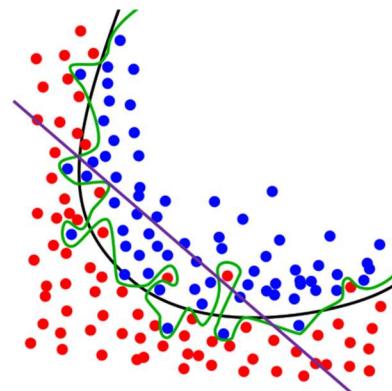
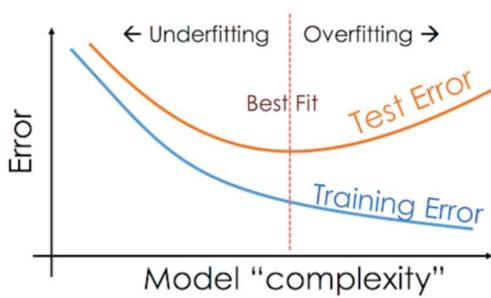
بایاس و واریانس

- بایاس اختلاف میان میانگین پیش‌بینی‌های مدل ما نسبت به مقادیر صحیح است
- واریانس میزان تغییرات پیش‌بینی مدل را نشان می‌دهد



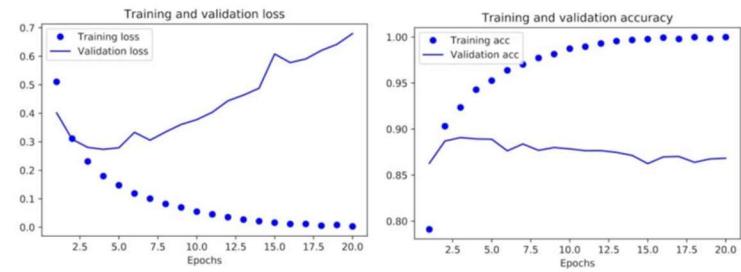
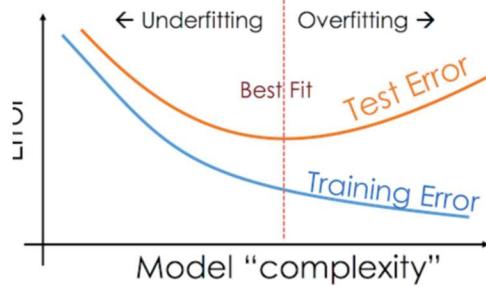
انتخاب مدل

- چگونه الگوهایی را کشف کنیم که تعمیم‌پذیر باشند؟
- چالش بزرگ این است که در زمان آموزش مدل فقط به مجموعه کوچکی از داده‌ها دسترسی داریم



انتخاب مدل

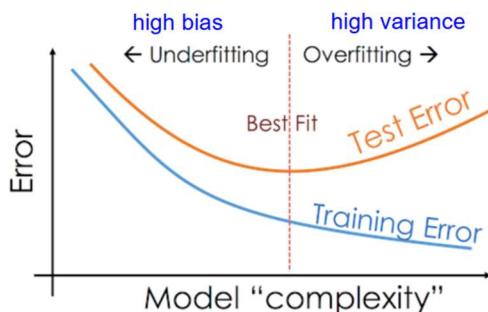
- عملکرد مدل‌ها بر روی داده‌های اعتبارسنجی که در آموزش مدل استفاده نشده‌اند معمولاً پس از چند دوره به اوج خود رسید و سپس شروع به تنزل می‌کند
- مدل به سرعت شروع به overfit شدن بر روی داده‌های آموزشی می‌کند
- **یادگیری نحوه مقابله با overfitting** یکی از نکات بسیار کلیدی در یادگیری ماشین است



بهینه‌سازی و تعمیم‌دهی

- بهینه‌سازی به تعیین پارامترهای مدل برای به دست آوردن بهترین عملکرد ممکن در داده‌های آموزشی (یادگیری در ML) اشاره دارد optimization > training

- تعمیم‌دهی به نحوه عملکرد مناسب مدل آموزش دیده بر روی داده‌هایی که تا کنون مشاهده نکرده است اشاره دارد generalization > test



- هدف دستیابی به تعمیم‌دهی مناسب است
 - اما کنترلی بر روی تعمیم‌دهی نداریم!
 - تنها می‌توانیم بر اساس داده‌های آموزشی پارامترهای مدل را تعیین کنیم

بهینه‌سازی و تعمیم‌دهی

- در ابتدای آموزش، بهینه‌سازی و تعمیم‌دهی با هم کاملاً مرتبط هستند
 - به مدل گفته می‌شود underfit است
 - شبکه هنوز تمام الگوهای مرتبط با مسئله مورد نظر در داده‌های آموزشی را یاد نگرفته است
- پس از چند تکرار، بهبود تعمیم‌دهی متوقف می‌شود و سپس شروع به تنزل می‌کند
 - مدل شروع به overfit شدن می‌کند



بیش برازش و کم برازش دو مشکل رایجی هستند که در هنگام آموزش شبکه‌های عصبی رخ می‌دهند.

بیش برازش (Overfitting) زمانی است که شبکه عصبی داده‌های آموزشی را خیلی خوب یاد می‌گیرد و قادر به تعمیم به داده‌های جدید نیست. این می‌تواند زمانی اتفاق بیفتد که شبکه بیش از حد پیچیده باشد، پارامترهای زیادی داشته باشد یا برای مدت طولانی آموزش داده شود. هنگامی که یک شبکه عصبی بیش از حد برازش می‌کند، در داده‌های آموزشی خوب عمل می‌کند اما در داده‌های آزمایش ضعیف عمل می‌کند.

کم برازش (Underfitting) بر عکس overfitting است. زمانی است که شبکه عصبی داده‌های آموزشی را به اندازه کافی یاد نمی‌گیرد. این می‌تواند زمانی اتفاق بیفتد که شبکه بیش از حد ساده باشد، پارامترهای کافی نداشته باشد یا برای مدت طولانی آموزش داده نشود. هنگامی که یک شبکه عصبی ضعیف می‌شود، هم در داده‌های آموزشی و هم در داده‌های آزمایشی عملکرد ضعیفی خواهد داشت.

در اینجا یک قیاس برای کمک به درک بیش از حد و عدم تناسب وجود دارد:

تصور کنید در حال تلاش برای یادگیری تشخیص انواع گل ها هستید. شما می توانید این کار را با دیدن تصاویر گل ها و تلاش برای به خاطر سپردن ویژگی های آنها انجام دهید. این مانند آموزش یک شبکه عصبی است.

اگر فقط به چند عکس از گل ها نگاه کنید، نمی توانید یاد بگیرید که چگونه همه آنها را تشخیص دهید. این مانند کم برازش یک شبکه عصبی است.

اگر به تعداد زیادی عکس از گل ها نگاه کنید، می توانید یاد بگیرید که چگونه بیشتر آنها را تشخیص دهید. با این حال، اگر به تصاویر بیش از حد نگاه کنید، ممکن است به جای ویژگی های گل ها، خود تصاویر را به خاطر بسیارید. این مانند نصب بیش از حد یک شبکه عصبی است.

هدف این است که تعادلی بین بیش برازش و کم برازش پیدا کنید. شما می خواهید شبکه عصبی را برای مدت زمان کافی آموزش دهید تا بتواند داده های آموزشی را به خوبی یاد بگیرد، اما نه آنقدر طولانی که بیش از حد برازنده شود.

تعدادی تکنیک وجود دارد که می توان از آنها برای جلوگیری از برازش بیش از حد استفاده کرد، مانند:

- استفاده از شبکه کوچکتر با پارامترهای کمتر
- آموزش برای دوره های کمتر
- استفاده از تکنیک های منظم سازی L1 یا L2
- استفاده از مجموعه اعتبارسنجی برای نظارت بر عملکرد شبکه بر روی داده های دیده نشده

اگر نگران بیش از حد برازش هستید، ایده خوبی است که از یک مجموعه اعتبارسنجی (validation set) برای نظارت بر عملکرد شبکه در داده های دیده نشده استفاده کنید. پس از آن می توانید آموزش شبکه را هنگامی که شروع به بیش از حد برازش در مجموعه اعتبار سنجی کرد، متوقف کنید.

کم برازش از بیش برازش مشکل است، زیرا اغلب می توان با آموزش شبکه برای مدت طولانی تر یا استفاده از یک شبکه پیچیده تر، آن را ببرطرف کرد. با این حال، مهم است که از کم برازش آگاهی داشته باشد تا در صورت وقوع، بتوانید آن را تشخیص دهید.

:chat gpt به بیان

Overfitting and underfitting are common challenges in machine learning, including in the context of neural networks. These problems occur when a model either performs too well on the training data but poorly on unseen data (overfitting) or when the model is too simple to capture the underlying patterns in the data (underfitting). Here's a detailed explanation of each:

1. Overfitting:

Overfitting happens when a model learns the training data too well, including the noise or random fluctuations in the data. This can lead to a highly complex model that doesn't generalize well to new, unseen data. Essentially, the model memorizes the training data rather than learning the underlying patterns. Signs of overfitting include excessively low training error but high test error or validation error.

To address overfitting, one can employ techniques such as:

- Regularization methods (e.g., L1 or L2 regularization)
- Cross-validation to tune hyperparameters
- Dropout layers, which randomly deactivate some neurons during training

- Early stopping, which stops training once the model performance on the validation set starts to degrade

2. Underfitting:

Underfitting occurs when a model is too simple to capture the underlying patterns in the data, leading to poor performance on both the training and test data. This can happen when the model is not complex enough to learn the relationships within the data.

To mitigate underfitting, one can consider:

- Increasing the model's complexity by adding more layers or neurons
- Using more advanced architectures or models
- Feature engineering to provide the model with more relevant information

Balancing the trade-off between overfitting and underfitting is crucial for creating a model that can generalize well to new, unseen data. This is often achieved through experimentation, tuning of hyperparameters, and the use of techniques that help in achieving the right balance between model complexity and generalizability.

(ب)

اگر یک مدل آموزش دیده دارد و می خواهد ارزیابی کنید که آیا مشکل بیش برآذش(overfitting) دارد، می توانید از تکنیک ها و معیارهای مختلفی برای ارزیابی عملکرد آن استفاده کنید. در اینجا چند روش متداول برای تعیین بیش برآذش پس از آموزش یک مدل آورده شده است:

۱. اعتبار سنجی مقابله (Cross-validation): استفاده از تکنیک های مانند اعتبارسنجی متقطع k-fold می تواند به ارزیابی عملکرد مدل در زیر مجموعه های مختلف داده ها کمک کند. اگر مدل در مقایسه با داده های اعتبارسنجی به طور قابل توجهی روی داده های آموزشی بهتر عمل کند، ممکن است بیش از حد برآذش داشته باشد.
۲. منحنی های یادگیری (Learning curves): ترسیم منحنی های یادگیری برای داده های آموزشی و اعتبارسنجی می تواند بینشی در مورد رفتار مدل ارائه دهد. اگر شکاف قابل توجهی بین منحنی های آموزشی و اعتبارسنجی وجود داشته باشد، نشان دهنده overfitting است.
۳. عملکرد مجموعه اعتبارسنجی (Validation set performance): مقایسه عملکرد مدل بر روی داده های آموزشی با عملکرد آن در یک مجموعه داده اعتبار سنجی جداگانه می تواند بیش از حد برآذش را نشان دهد. اگر مدل در مقایسه با داده های اعتبارسنجی به طور قابل توجهی روی داده های آموزشی بهتر عمل کند، ممکن است بیش برآذش وجود داشته باشد.
۴. آزمایش بر روی داده های دیده نشده (Testing on unseen data): ارزیابی عملکرد مدل بر روی مجموعه داده های آزمایشی کاملاً دیده نشده می تواند نشانه روشی از تطابق بیش از حد مدل با داده های آموزشی ارائه دهد. اگر عملکرد مدل به طور قابل توجهی در مجموعه تست در مقایسه با مجموعه آموزشی کاهش یابد، نشان دهنده بیش برآذش است.
۵. تکنیک های منظم سازی (Regularization techniques): می توانید بررسی کنید که آیا استفاده از تکنیک های منظم سازی در طول تمرین، مانند Dropout یا منظم سازی L1/L2، به جلوگیری از overfitting کمک کرده است یا خیر. اگر مدل با گنجاندن این تکنیک ها عملکرد بهتری داشته باشد، نشان می دهد که بیش برآذش ممکن است بدون آنها مشکل ساز بوده باشد.

با استفاده از این روش ها، می توانید درک بهتری از اینکه آیا مدل آموزش دیده شما بیش برآذش دارد یا خیر و اگر چنین است، چگونه آن را برای دستیابی به عملکرد تعمیم بهتر تنظیم کنید، به دست آورید.

جواب سوال به بیان chat gpt

There are a few ways to determine the overfitting of a trained model:

1. Compare the performance of the model on the training data to the performance of the model on a held-out test set. If the model performs significantly better on the training data than on the test set, then the model is likely overfitting.
2. Plot the training and validation loss curves. The training loss curve will typically decrease as the model is trained, while the validation loss curve will typically plateau or even increase if the model is overfitting.
3. Use regularization techniques. Regularization techniques such as L1 and L2 regularization penalize the model for having complex weights. This can help to prevent the model from overfitting.
4. Use early stopping. Early stopping is a technique where the model training is stopped early if the validation loss starts to increase. This can help to prevent the model from overfitting.

Here are some specific steps you can follow to determine the overfitting of a trained model:

1. Split your data into a training set and a test set.
2. Train your model on the training set.
3. Evaluate the model's performance on the test set.
4. Compare the model's performance on the training set to the model's performance on the test set.
5. If the model performs significantly better on the training set than on the test set, then the model is likely overfitting.

If your model is overfitting, you can try to reduce overfitting by using regularization techniques, early stopping, or by collecting more training data.

Here are some additional tips for preventing overfitting:

- Avoid using too complex of a model.
- Use a validation set to monitor the performance of your model on unseen data.
- Use regularization techniques to penalize the model for having complex weights.
- Use early stopping to stop training the model early if the validation loss starts to increase.

(پ)

Dropout

باعث میشه نورون ها با هم نتونن به الگوی خاصی را حفظ کن و بیشتر تعمیم دهی کنن

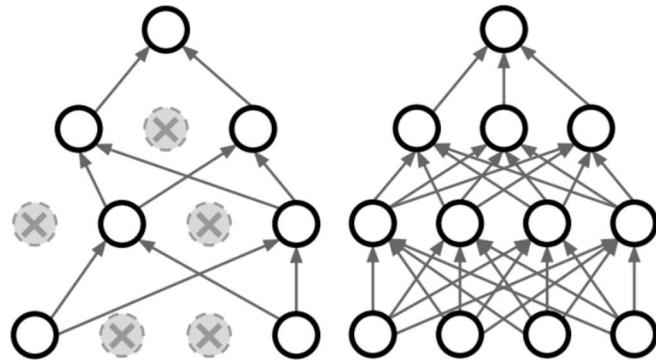
$$\text{unit}(i,j) = 0$$

- در هر تکرار، به صورت تصادفی مقدار تعدادی نورون را صفر می‌کند

- مشابه با نویز ضرب‌شونده با مقادیر باینری است

- احتمال حذف هر واحد یک ابرپارامتر است

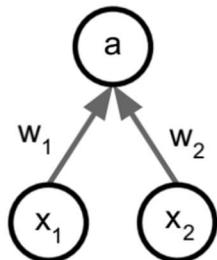
- مقدار 0.5 متداول است



Dropout

$$y = f(x) = E_z[f(x, z)] = \sum_z p(z)f(x, z) \quad \text{• یک نورون ساده با دو ورودی را در نظر بگیرید:}$$

$$\begin{aligned} E[a] &= p^2(w_1x_1 + w_2x_2) + p(1-p)(w_1x_1 + w_20) \\ &\quad + (1-p)p(w_10 + w_2x_2) + (1-p)^2(w_10 + w_20) \\ &= p(w_1x_1 + w_2x_2) \end{aligned}$$



- در زمان تست، تمام نورون‌ها فعال هستند اما خروجی هر نورون را در ضریب p ضرب می‌کنیم

Dropout

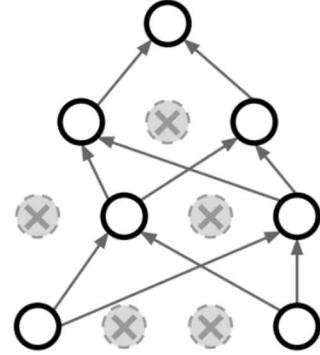
```

p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) # first dropout mask.
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) # second dropout mask.
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3      drop in forward pas

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # scale the activations
    out = np.dot(W3, H2) + b3
    scale at test time

```



Inverted Dropout

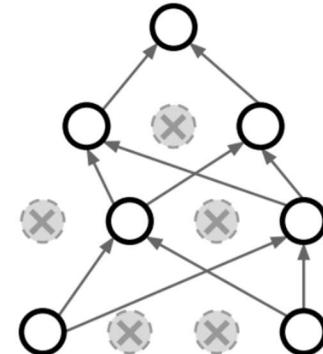
```

p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3      drop in forward pass

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
    test time is unchanged

```



همان طور که در بالا مبینیم برای اعمال Dropout دو روش وجود دارد. یک Dropout معمولی که در زمان تست با ضرب p مقادیر را اصلاح می کند. و دیگری Inverted Dropout که در زمان آموختن با تقسیم کردن بر p مقادیر را اصلاح می کند. ما در اینجا روش معمولی را در نظر میگیریم و با فرض آن پیش می رویم.

با خروج اینکه Dropout معنولی داریم.

اگر در مرحله آموزش باسیم: ماسک Dropout را در خروجی لایه شبکه عصبی

ضربی کنیم و می را در زمان تست لحاظ می کنیم.

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \star \begin{bmatrix} 1.6 & -0.7 & -0.2 & 1.9 \\ -2.3 & 2.5 & 2.5 & -0.9 \\ -0.5 & 3.2 & 3.7 & -0.4 \\ 1.3 & -0.4 & -2.6 & 1.2 \end{bmatrix} = \begin{bmatrix} 1.6 & 0 & 0 & 1.9 \\ 0 & 2.5 & 2.5 & 0 \\ 0 & 3.2 & 3.7 & 0 \\ 1.3 & 0 & 0 & 1.2 \end{bmatrix}$$

Output \star Dropoutmask = train result

1.6	0	0	1.9
0	2.5	2.5	0
0	3.2	3.7	0
1.3	0	0	1.2

= مقدار بینایی لایه بعد از اعمال
Dropout
در زمان آموزش

اگر در مرحله تست باسیم: ماسک Dropout در خروجی لایه شبکه عصبی ضرب

نهایی شود و می خودی هادر ρ ضربی سوند تا مقادیر اصلاح شوند.

$$\rho = \frac{\text{تعداد 1 ها}}{\text{تعداد 1 ها} + \text{تعداد 0 ها}} = \frac{8}{8+8} = \frac{8}{16} = \frac{1}{2}$$

$$\begin{bmatrix} 1.6 & -0.7 & -0.2 & 1.9 \\ -2.3 & 2.5 & 2.5 & -0.9 \\ -0.5 & 3.2 & 3.7 & -0.4 \\ 1.3 & -0.4 & -2.6 & 1.2 \end{bmatrix} \times \frac{1}{2} = \begin{bmatrix} 0.8 & -0.35 & -0.1 & 0.95 \\ -1.15 & 1.25 & 1.25 & -0.45 \\ -0.25 & 1.6 & 1.85 & -0.2 \\ 0.65 & -0.2 & -1.3 & 0.6 \end{bmatrix}$$

مقدار بینایی
لایه بعد از
اعمال Dropout
در زمان
تست

Output $\times \rho$ = test result

مراجع:

<https://chat.openai.com/>
<https://bard.google.com/>
<https://claude.ai/chats>

سوال دوم

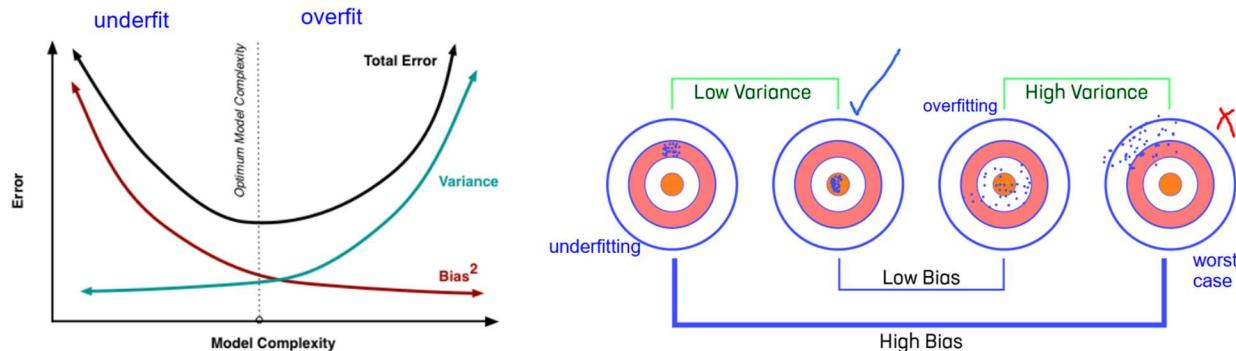
(الف)

بایاس و واریانس یک مدل KNN با تغییر مقدار K تغییر می کند.



بایاس و واریانس

- بایاس اختلاف میان میانگین پیش‌بینی‌های مدل ما نسبت به مقادیر صحیح است
- واریانس میزان تغییرات پیش‌بینی مدل را نشان می‌دهد

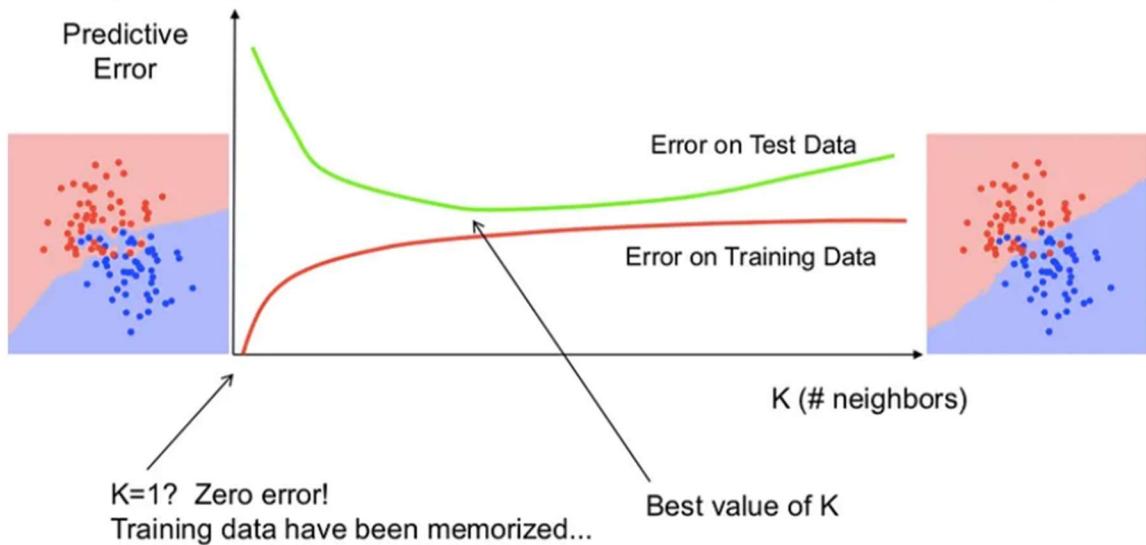


بایاس(سوگیری) تمایل یک مدل به پیش‌بینی‌هایی است که به طور مداوم خیلی زیاد یا خیلی پایین هستند. واریانس تمایل یک مدل به پیش‌بینی‌های مختلف برای ورودی یکسان، بسته به داده‌های آموزشی مورد استفاده است.

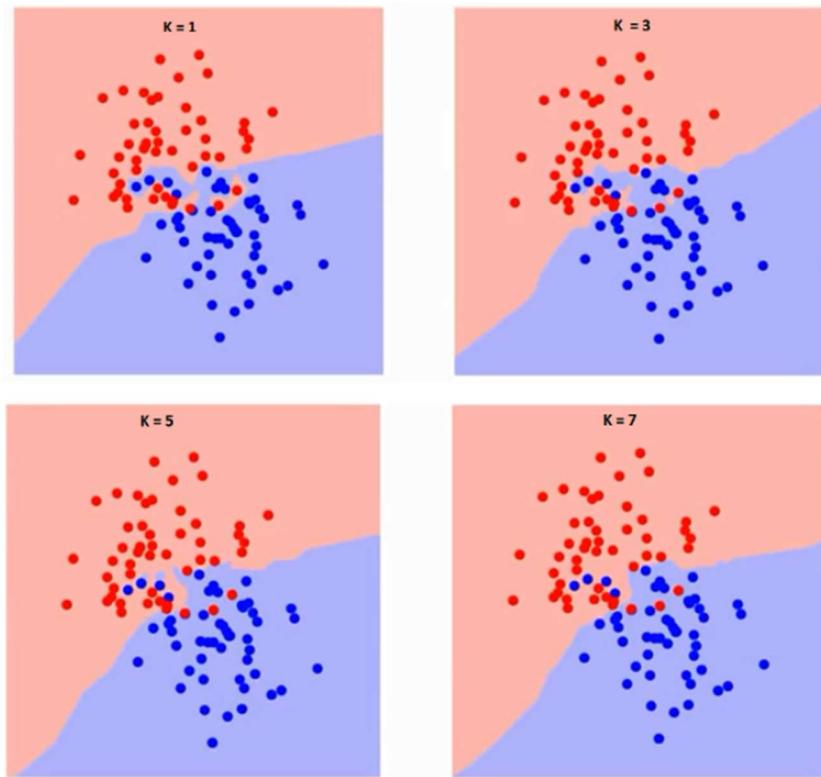
به طور کلی، با افزایش K ، بایاس یک مدل KNN افزایش می‌یابد و واریانس کاهش می‌یابد. در واقع، خطای آموزش افزایش می‌یابد (افزایش بایاس)، اما خطای آزمون ممکن است در همان زمان کاهش یابد(کاهش واریانس). این به این دلیل است که K بزرگتر به این معنی است که مدل نسبت به نقاط داده بیشتری میانگین می‌گیرد، که تأثیر نقاط داده جدآگانه را کاهش می‌دهد و مدل را در برابر نویز قوی ترمی کند. با این حال، همچنین به این معنی است که مدل کمتر احتمال دارد جزئیات دقیق‌تری از داده‌ها را بگیرد، که می‌تواند منجر به بایاس(سوگیری) شود. همچنین با کاهش K ، واریانس یک مدل KNN افزایش می‌یابد و بایاس کاهش می‌یابد. مثلاً، یک حالت شدید، $K=1$ را در نظر بگیرید، چه اتفاقی خواهد افتاد؟ داده‌های آموزشی کاملاً درست پیش‌بینی می‌شوند. بایاس زمانی . خواهد بود که $K=1$ ، با این حال، زمانی که به داده‌های جدید (در مجموعه آزمایشی) می‌رسد، شанс بیشتری برای خطای دارد، که باعث واریانس بالا می‌شود.

نمودارهای زیر که در لینک‌هایی که در مراجع قرار داده ام این موضوع را تایید می‌کنند.

Error rates and K

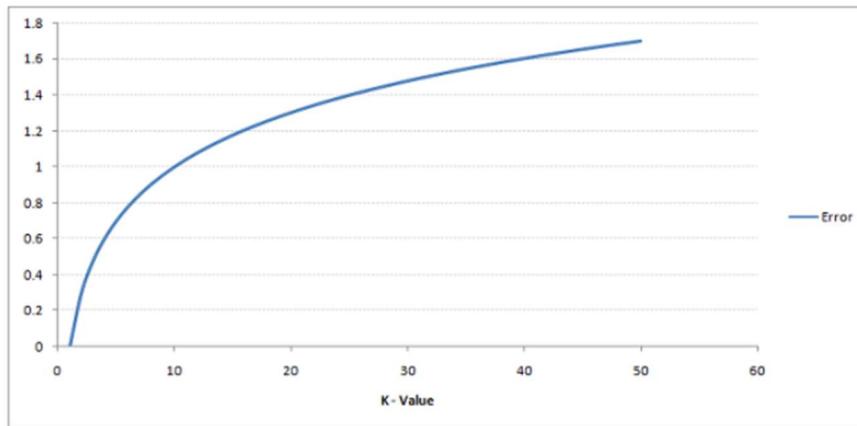


Error rate and K. Credit: <http://sameersingh.org/courses/gml/fa17/sched.html>

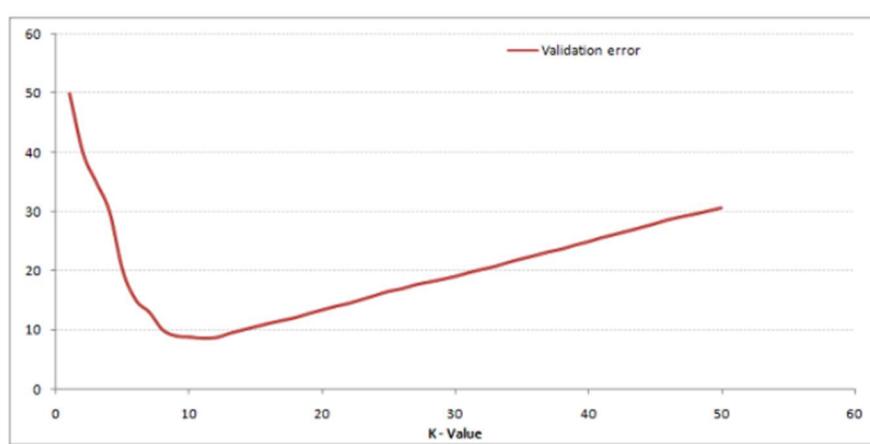


If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority. The training error

increasing to infinity it finally becomes all blue or all red depending on the total majority. The training error rate and the validation error rate are two parameters we need to access different K-value. Following is the curve for the training error rate with a varying value of K :



As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself.Hence the prediction is always accurate with K=1. If validation error curve to any training data point is itself.Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K:



This makes the story more clear. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increase with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K. This value of K should be used for all predictions.

در اینجا یک قیاس برای کمک به درک رابطه بین سوگیری و واریانس در KNN وجود دارد:
تصویر کنید که در حال تلاش برای تخمین قد متوسط افراد در یک اتاق هستید. می توانید این کار را با پرسیدن قد چند نفر و میانگین پاسخ ها انجام دهید. این مانند استفاده از یک مدل KNN با مقدار کمی K است.

اگر فقط از چند نفر (تعداد کم) قد آنها را پرسید، تخمین شما احتمالاً تغییرات زیادی دارد، زیرا به شدت تحت تأثیر قد افرادی است که از آنها پرسیده اید. در واقع، احتمالاً واریانس نسبتاً بالای خواهیم داشت، زیرا اگر از افراد متفاوتی پرسید، بسته به افراد مختلف (مثلاً همه آنها بسکتبالیست باشند) تغییر زیادی کند.

اگر از تعداد زیادی از افراد قد آنها را پرسید، تخمین شما احتمالاً تغییرات کمتری خواهد داشت و به سما میانگین کل سوگیری کند، زیرا میانگین آنها نماینده کل جامعه است. در واقع، بایاس بیشتری خواهد داشت، چون تعداد افراد زیاد است، میانگین قد نمی تواند به طور قابل توجهی تغییر کند.

همین اصل در مورد مدل های KNN نیز صدق می کند. مقدار کوچک K منجر به مدلی با واریانس بیشتر اما بایاس کمتر می شود، در حالی که مقدار زیاد K منجر به مدلی با بایاس بیشتر اما واریانس کمتر می شود.

هنگام انتخاب مقدار K برای یک مدل KNN، مهم است که مبادله بین بایاس و واریانس را در نظر بگیرید. اگر داده ها نویز دارند، مهم است که مقدار K بزرگتری را برای کاهش واریانس انتخاب کنید. با این حال، اگر داده ها تمیز هستند، مهم است که مقدار K کوچکتری را برای کاهش بایاس انتخاب کنید.

در اینجا چند نکته برای انتخاب مقدار K برای مدل KNN وجود دارد:

- از اعتبارسنجی متقارط (cross-validation) برای ارزیابی عملکرد مدل برای مقادیر مختلف K استفاده کنید.
- مقداری از K را انتخاب کنید که خطای در مجموعه اعتبارسنجی را به حداقل برساند.
- اگر داده ها دارای نویز هستند، مقدار K بزرگتری را برای کاهش واریانس انتخاب کنید.
- اگر داده ها تمیز هستند، مقدار K کوچکتری را برای کاهش بایاس انتخاب کنید.
- جواب سوال به بیان chat gpt

In the K-Nearest Neighbors (KNN) algorithm, the value of K refers to the number of nearest neighbors that are considered when making a prediction. Adjusting the value of K can significantly impact the bias and variance of the model.

1. **Bias:** Bias refers to the error that is introduced by approximating a real-world problem with a much simpler model. In the context of KNN, the bias tends to increase with increasing K . This is because as K increases, the model considers more data points for decision-making, leading to a smoother decision boundary and a reduction in the tendency to overfit the training data. Consequently, higher values of K generally result in a higher bias.
2. **Variance:** Variance refers to the variability of model predictions for a given data point. In the case of KNN, variance tends to increase with lower values of K . This is because a lower K implies a smaller neighborhood is considered, potentially introducing more noise into the decision-making process. With smaller neighborhoods, the decision boundary becomes more complex and sensitive to small changes in the data, leading to a higher variance.

Therefore, the relationship between the value of K and bias-variance can be summarized as follows:

- **Low K ($K=1$):** Low bias, high variance. The model closely follows the training data, which can result in overfitting and high variance, making it sensitive to noise.
- **Moderate K ($2 < K < \sqrt{n}$, n being the total number of data points):** A balance between bias and variance. The model's decision boundary becomes smoother, reducing overfitting and improving generalization.

- **High K ($K > \sqrt{n}$):** High bias, low variance. The model becomes more robust, but it might oversimplify the underlying patterns in the data, leading to increased bias.

Selecting an appropriate value of K is crucial to achieve the right balance between bias and variance. This choice often involves the use of cross-validation techniques and model evaluation metrics to find the optimal value of K that leads to the best generalization performance on unseen data.

(ب)

- استفاده از منظم سازی، ممکن است باعث تضعیف عملکرد مدل شود.
درست

منظم سازی تکنیکی است که برای جلوگیری از بیش برآذش در مدل های یادگیری ماشین استفاده می شود. با جریمه کردن مدل برای داشتن وزن های بزرگ و پیچیده کار می کند. این تکنیک گاهی اوقات می تواند منجر به کاهش عملکرد در داده های آموخته شود، اما معمولاً عملکرد داده های تست را بهبود می بخشد. ولی چون در گزاره ذکر شده است ممکن است مثلاً حالت خاص و نه عمومی) را در نظر بگیرید که واقعاً داده های پیچیده هستند و نیاز داریم وزن های بزرگ و پیچیده باشند لذا در چنین حالتی ممکن است باعث تضعیف عملکرد مدل شود و اگر از آن استفاده نمی کردیم عملکرد بهتری داشتیم. همچنین اگر ضریب منظم سازی به درستی انتخاب نشود و خیلی بزرگ باشد مدل بیش از حد ساده خواهد شد که منجر به کاهش قدرت مدل و پیچیدگی آن و در نهایت تضعیف عملکرد مدل خواهد شد.

تعريف منظم سازی: منظم سازی با اضافه کردن یک عبارت جریمه به تابع ضرر کار می کند. این عبارت جریمه مدل را برای داشتن وزن های پیچیده جریمه می کند. هر چه وزن های مدل پیچیده تر باشد، مجازات بیشتر خواهد بود. این مدل را مجبور می کند تا عملکرد ساده تری را یاد بگیرد، که می تواند به جلوگیری از برآذش بیش از حد کم کند.

با این حال، منظم سازی همچنین می تواند عملکرد مدل را در داده های آموخته کاهش دهد. این به این دلیل است که منظم سازی اساساً مدل را مجبور می کند تا برخی از اطلاعات موجود در داده های آموخته را نادیده بگیرد. اگر داده های آموخته کم باشد، این می تواند منجر به کاهش قابل توجهی در عملکرد شود.

- اضافه کردن تعداد زیاد ویژگی های جدید، باعث جلوگیری از بیش برآذش می شود.
غلط

افزودن ویژگی های جدید می تواند به بهبود عملکرد یک مدل یادگیری ماشینی کمک کند، اما اگر ویژگی ها به دقت انتخاب نشده باشند، می تواند منجر به بیش برآذش شود. افزودن تعداد زیادی از ویژگی های جدید می تواند خطر افزایش بیش برآذش افزایش دهد، بهویژه اگر ویژگی ها correlated یا اضافی باشند. به بیان دیگر، افزودن تعداد زیادی از ویژگی های جدید بدون انتخاب یا مهندسی مناسب ویژگی ها می تواند منجر به بیش برآذش شود. مدل ممکن است شروع به بخاطر سپردن نویز و الگوهای نامربوط در داده ها کند (نویز و الگوهای نا مرتبط را حفظ کند) که در نتیجه تعمیم ضعیفی به نقاط داده جدید ایجاد می کند. بسیار مهم است که به جای افزودن ویژگی های بیش از حد یا اضافی، فقط ویژگی های مرتبط را اضافه کنید که به ثبت الگوهای اساسی داده ها کمک می کند. پس گزاره غلط است و اضافه کردن تعداد زیاد ویژگی های جدید، ممکن است باعث بیش برآذش شود.

- با زیاد کردن ضریب منظم سازی، احتمال بیش برآذش بیشتر می شود.
غلط

ضریب منظم سازی پارامتری است که میزان جریمه شدن مدل را برای داشتن وزن های بزرگ و پیچیده کنترل می کند. افزایش ضریب منظم سازی مدل را محافظه کارتر می کند و احتمال بیش برآذش را کمتر می کند. با این حال، اگر ضریب منظم سازی بیش از حد بالا تنظیم شود، مدل ممکن است نتواند داده های آموخته را به خوبی یاد بگیرد، که می تواند منجر به underfitting شود. در حقیقت، افزایش ضریب منظم سازی به طور موثر جریمه مدل های پیچیده را افزایش می دهد و مدل های ساده تر را در طول فرآیند آموختش تشویق می کند. با انجام این کار، مدل کمتر به داده های آموخته بیش برآذش می شود زیرا از یادگیری نویز یا نوسانات کوچک جلوگیری می کند. بنابراین، افزایش ضریب تنظیم به طور کلی با محدود کردن پیچیدگی مدل به کاهش بیش از حد برآذش کمک می کند و گزاره غلط است.

(ب)

منظم سازی پارامتر L2

$$\Omega(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$\tilde{L}(\mathbf{w}, b) = L(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

- اگر بجای L2 از فاصله اقلیدوی استفاده کنیم محاسبه مشتق پیچیده می شود

$$\nabla_{\mathbf{w}} \tilde{L}(\mathbf{w}, b) = \nabla_{\mathbf{w}} L(\mathbf{w}, b) + \lambda \mathbf{w}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta (\lambda \mathbf{w} + \nabla_{\mathbf{w}} L(\mathbf{w}, b))$$

$$\mathbf{w} \leftarrow (1 - \eta \lambda) \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}, b)$$

- به این روش، Weight Decay هم گفته می شود

کاهش اندازه پارامتر

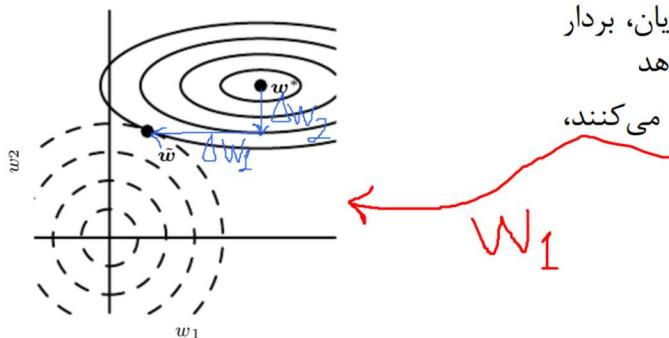
منظم سازی پارامتر L2

$$\Omega(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$\mathbf{w} \leftarrow (1 - \eta \lambda) \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}, b)$$

- قبل از انجام بهروزرسانی معمولی مبتنی بر گرادیان، بردار وزن را در هر گام با یک ضریب ثابت کاهش می دهد

- وزن هایی که تغییر کمتری در تابع ضرر ایجاد می کنند، اهمیت کمتری دارند و بیشتر کاهش می یابند



منظمسازی پارامتر L1

- منظمسازی L1 بر روی پارامترهای شبکه \mathbf{w} به صورت زیر تعریف می‌شود

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_i |w_i|$$

$$\tilde{L}(\mathbf{w}, b) = L(\mathbf{w}, b) + \lambda \|\mathbf{w}\|_1$$

$$\nabla_{\mathbf{w}} \tilde{L}(\mathbf{w}, b) = \nabla_{\mathbf{w}} L(\mathbf{w}, b) + \lambda \text{sign}(\mathbf{w})$$

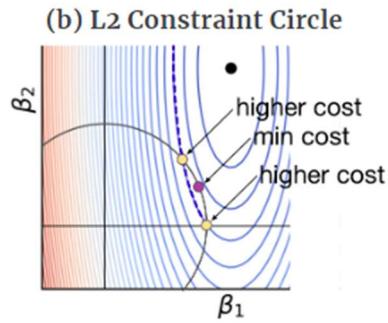
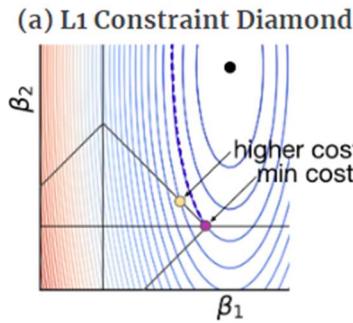
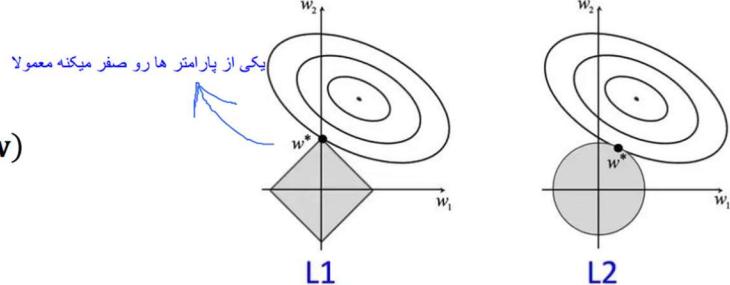


Figure 3.2. L1 and L2 constraint regions get different coefficient locations, on the diamond and circle, for the same loss function. Keep in mind that there are an infinite number of contour lines and there is a contour line exactly meeting the L2 purple dot.

از نقطه نظر عملی، L1 تمایل دارد ضرایب را به صفر کاهش دهد در حالی که L2 تمایل دارد ضرایب را به طور مساوی کاهش دهد. بنابراین L1 برای انتخاب ویژگی (feature selection) مفید است، زیرا می‌توانیم متغیرهای مرتبط با ضرایب را که به صفر می‌رسند حذف کنیم. از سوی دیگر، L2 زمانی مفید است که ویژگی‌های هم خطی/همبسته (collinear/codependent) داشته باشید.

بر اساس مقادیر وزن ارائه شده، در اینجا تجزیه و تحلیل من در هر آزمایش را میبینید:

- Wexp1: تنظیم L2 احتمالاً در اینجا استفاده شده است. وزن‌ها همگی بسیار کوچک و از نظر مقدار نزدیک هستند که نشانه انقباض (shrinkage) ناشی از L2 است.
- Wexp2: تنظیم L1 احتمالاً در اینجا استفاده شده است. یک وزن بزرگ است (۱) در حالی که بقیه . هستند، که پراکندگی (تنگی، sparsity) ناشی از تنظیم L1 را نشان می‌دهد.
- Wexp3: احتمالاً در اینجا از هیچ تنظیمی استفاده نشده است. وزن‌ها از نظر اندازه بسیار متفاوت هستند و هیچ کدام صفر نیستند، که نشانه ای از بیش برازش بدون منظم سازی (overfitting with no regularization) است.
- Wexp4: تنظیم L1 احتمالاً استفاده شده است. پراکندگی با برخی صفرها و غیرصفرها وجود دارد. اگر از L2 استفاده می‌شد، مقادیر کوچکتر و نزدیک تر بودند اما احتمال کمتری داشت که دقیقاً صفر باشند.

علائم کلیدی عبارتند از:

- L2: همه وزن ها به سمت صفر کوچک می شوند.
- L1: بسیاری از وزن ها دقیقاً صفر می شوند. (تنکی، پراکندگی، sparsity)
- بدون منظم سازی(regularization): وزن ها می توانند خودسرانه و خیلی متفاوت باشند و مقیاس های متفاوت داشته باشند.

بنابراین L2 و L1 نشانه های متفاوتی را در مقادیر وزن ایجاد می کنند که با آنچه در W_{exp1} , W_{exp2} و W_{exp4} می بینیم مطابقت دارد.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

<https://medium.com/30-days-of-machine-learning/day-3-k-nearest-neighbors-and-bias-variance-tradeoff-75f84d515bdb>

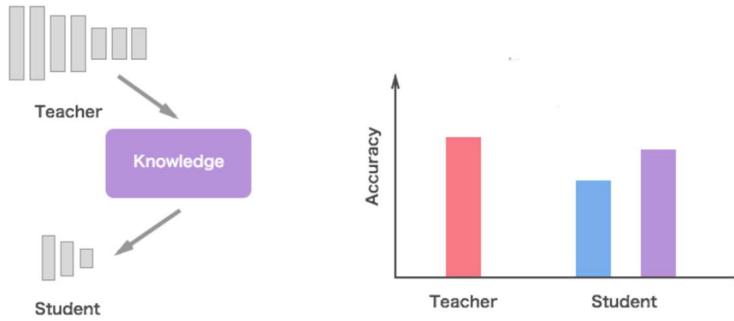
<https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>

<https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization>

<https://explained.ai/regularization/L1vsL2.html>

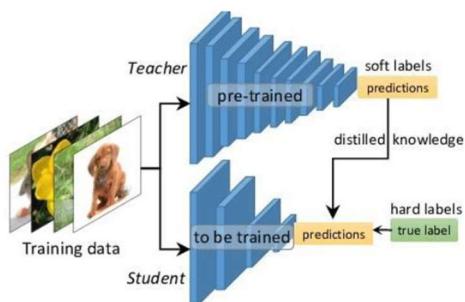
تقطیر دانش (Knowledge Distillation)

- در یادگیری ماشین، تقطیر دانش به فرآیندی گفته می‌شود که دانش از یک مدل بزرگ‌تر (معلم) به یک مدل کوچک‌تر (دانش‌آموز) منتقل می‌شود
 - یکی از کاربردهای آن توسعه مدل‌های سریع با دقت مناسب است



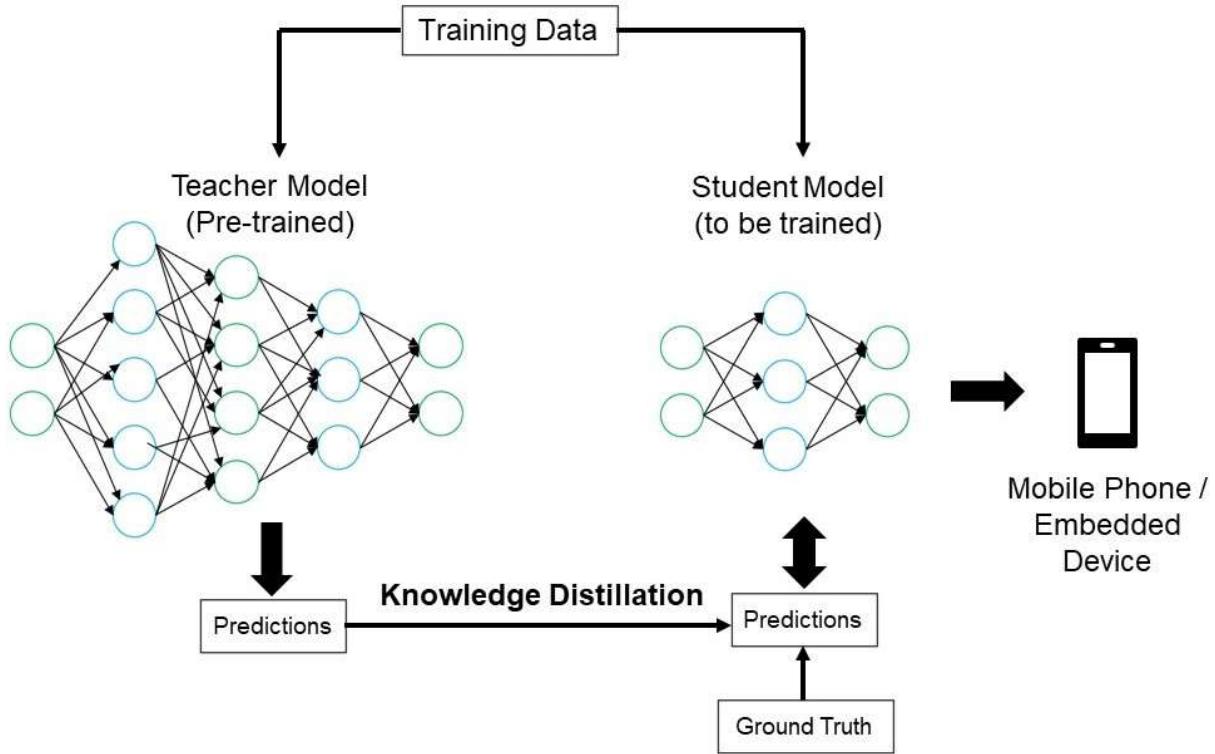
تقطیر دانش

- روش‌های مختلفی برای تقطیر دانش پیشنهاد شده است که یکی از آنها تقطیر در سطح پاسخ یا خروجی شبکه است
- از پیش‌بینی شبکه معلم برای نظارت بر پاسخ شبکه دانش‌آموز به عنوان برچسب‌های نرم استفاده می‌شود



(الف)

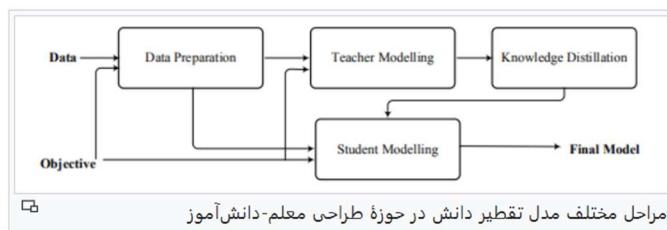
تقطیر دانش (به انگلیسی: Knowledge distillation) یکی از روش‌های مرسوم برای انتقال اطلاعات بین مدل‌ها است که در چند سال گذشته بسیار مورد بررسی قرار گرفته است. در دنیای یادگیری ماشین، به فرایند انتقال دانش از یک مدل نسبتاً بزرگ به یک مدل کوچک تقطیر دانش گفته می‌شود. مدل‌های بزرگ مثل شبکه‌های عصبی ژرف یا مدل‌های یادگیری ماشین جمعی (به انگلیسی: Ensemble Models) ظرفیت دانش و یادگیری بیشتری در مقایسه با مدل‌های کوچک‌تر دارند اما از لحاظ محاسباتی بسیار سنگین‌تر هستند. تقطیر دانش، این دانش را از مدل‌های پیچیده به مدل‌های ساده‌تر و کوچک‌تر منتقل می‌کند و تلاش می‌کند که این کار را با کمترین میزان هدرفت دانش انجام دهد.



در واقع نیاز به آن، زمانی حس شد که مدل‌های پیچیده‌ای داریم که توانایی حل مسائل سخت را دارند و می‌خواهیم آنها را در دستگاه‌های تلفن همراه یا جاها‌ی دیگر که کمبود منابع داریم، به کار بگیریم. به طور معمول در دنیای یادگیری ماشین، ما از مدل‌های مشابه برای آموزش و استنتاج استفاده می‌کنیم اما نکته قابل توجه این است که نیازمندی‌های این دو قسمت با هم متفاوت است. در یادگیری معمولاً ما دنبال این هستیم که پارامترهای یادگیری را بدست آوریم. برای این کار می‌توان زمان بسیار زیادی را صرف کرد و مسیرهای مختلف زیادی را امتحان کرد. همچنین می‌توان از تکنیک‌های مختلف استفاده کرد؛ مثلًا عمق شبکه را بیشتر کنیم یا از داده‌های بیشتری کمک بگیریم یا برای تنظیم مدل (به انگلیسی: Dropout) از حذف تصادفی (به انگلیسی: Regularization) استفاده کنیم. اما در استنتاج بسیار واضح است که به این کارها و امتحان کردن‌ها نیاز نداریم. در مرحله استنتاج مهم‌ترین نیازها دقت بالای مدل و زمان مصرفی برای پیش‌بینی کردن است. اگر مدل شما بسیار کند باشد یا منابع محاسباتی زیادی نیاز داشته باشد، دیگر دقت بالای آن اهمیت چندانی پیدا نمی‌کند. در اینجا است که ایده اساسی تقطیر دانش شکل می‌گیرد.

این روش، فرایندی برای آموزش یک مدل ساده است، به طوری که دانش مدل پیچیده را تا حد امکان به آن انتقال دهیم. معمولاً مدل پیچیده را معلم (به انگلیسی: Teacher Model) و مدل ساده را دانش‌آموز (به انگلیسی: Student Model) می‌نامیم. مدل معلم به طور معمول مدلی است که پارامترهای بسیار زیادی دارد یا یک مجموعه از چندین مدل است و نمی‌توان از آنها به راحتی در جاهای مختلف که توانایی محاسباتی بسیار زیادی نداریم، استفاده کرد. مدل دانش‌آموز نیز معمولاً مدلی ساده و بسیار سریع است.

نگاهی دقیق‌تر به تقطیر دانش [ویرایش]



یک راه برای انتقال دادن تعمیم‌سازی (به انگلیسی: Generalization) از مدل معلم به مدل دانش‌آموز این است که از توزیع احتمال خروجی مدل معلم که آن را اهداف نرم (به انگلیسی: Soft Targets) می‌نامیم، برای آموزش مدل دانش‌آموز استفاده کنیم. در طرف دیگر هم تعدادی داده همراه با برچسب داریم که آن‌ها را اهداف سخت (به انگلیسی: Hard Targets) می‌نامیم. از هر دوی آن‌ها در این فرایند تقطیر استفاده می‌شود. اگر مدل معلم مجموعه‌ای از مدل‌های ساده باشد آن‌گاه می‌توان از میانگین حسابی یا هندسی توزیع‌های خروجی به عنوان اهداف نرم استفاده کرد.^[۳] در واقع در اینجا به جای آموزش مستقیم مدل دانش آموز روی داده‌های خام، مدل سعی می‌کند تا خروجی شبکه معلم را تقلید کند. اگر آن‌تropy اهداف نرم زیاد باشد، اطلاعات بسیار زیادی را با واریانس کمتر در مقایسه با اهداف سخت متناظر، در دل خود دارند. از روی همین می‌توان فهمید که مدل دانش‌آموز در مقایسه با مدلی که به صورت عادی بخواهد آموزش ببیند، می‌تواند با حجم داده بسیار کمتر آموزش ببیند و حتی دچار مشکل بیش‌برازش (به انگلیسی: Overfitting) نشود. در مواردی حتی دیده شده است که وقتی مدل دانش‌آموز از معلم به دلیل این که تعمیم‌سازی بهتری و بیش‌برازش کمتری دارد، بیشتر است.

در شکل بالا مراحل مختلف مدل‌های مبتنی بر تقطیر دانش را می‌بینید. در قسمت اول جمع‌آوری داده، مورد نیاز است. در بسیاری از کاربردها، شبکه‌های معلم و دانش‌آموز به طور مستقیم بر روی داده خام نمی‌توانند آموزش ببینند؛ بنابراین یک سری تبدیلات بر روی داده‌ها ممکن است نیاز شود. در مرحله بعدی که طراحی مدل معلم است، ساختار مدل بر اساس اهداف و مسئله تعیین می‌گردد. همچنین تعداد معلم‌ها (اگر قرار است چند شبکه معلم داشته باشیم که هر یک روی بخشی از داده کار کنند) و نحوه ارتباط آن‌ها با همدیگر تعیین می‌گردد.

مرحله بعدی تقطیر دانش است که هسته کلی کار است. در این مرحله معلوم می‌شود که تقطیر به چه شکل صورت بگیرد. تعیین می‌کنید که روش انتقال به چه شکل باشد. این که از اهداف سخت یا اهداف نرم یا هر دو استفاده کنیم و تقطیر تنها از انتهاهای مدل معلم یا لایه‌های میانی صورت گیرد نیز در این قسمت معلوم می‌شوند. در نهایت در مرحله طراحی مدل دانش‌آموز، ساختار این شبکه و این که چطور اطلاعات را از معلم دریافت کند، مورد بررسی قرار می‌گیرد. نکته قابل توجه این است که این بخش به صورت مستقیم به داده و هدف متصل است، زیرا مدل دانش‌آموز بر اساس نوع مسئله و داده‌های مورد بررسی طراحی می‌شود.

:chat gpt به بیان

Knowledge distillation is a technique used to transfer knowledge from a large and complex model (the teacher model) to a smaller and simpler model (the student model). It is often used to compress large models so that they can be deployed on devices with limited resources, such as smartphones and edge devices.

The knowledge distillation process works by training the student model to mimic the outputs of the teacher model. This is done by minimizing a loss function that measures the difference between the outputs of the two models.

There are two main types of knowledge distillation:

- Response-based knowledge distillation: This type of knowledge distillation minimizes the difference between the final predictions of the teacher and student models.
- Feature-based knowledge distillation: This type of knowledge distillation minimizes the difference between the intermediate representations of the teacher and student models.

Knowledge distillation has been shown to be effective in improving the performance of small models on a variety of tasks, including image classification, natural language processing, and speech recognition.

Here are some of the benefits of using knowledge distillation:

- It can be used to compress large models so that they can be deployed on devices with limited resources.

- It can improve the performance of small models.
- It can be used to regularize large models and prevent overfitting.
- It can be used to transfer knowledge from one model to another, even if the two models are trained on different datasets or have different architectures.

Here are some of the applications of knowledge distillation:

- Model compression: Knowledge distillation is often used to compress large models so that they can be deployed on devices with limited resources. For example, knowledge distillation can be used to compress a large image classification model so that it can be deployed on a smartphone.
- Transfer learning: Knowledge distillation can be used to transfer knowledge from one model to another, even if the two models are trained on different datasets or have different architectures. For example, knowledge distillation can be used to transfer knowledge from a large language model trained on general text to a smaller language model trained on a specific domain, such as medical text or legal text.
- Model regularization: Knowledge distillation can be used to regularize large models and prevent overfitting. This is done by training the student model to mimic the outputs of the teacher model, which forces the student model to learn a more robust representation of the data.

(ب)

معماری شکل سوال یک معماری تقطیر دانش است که از شبکه معلم برای آموزش شبکه دانش آموز استفاده می کند. شبکه دانش آموز برای تقلید از خروجی های شبکه معلم آموزش دیده است که دانش را از شبکه معلم به شبکه دانش آموز منتقل می کند. در واقع، از تقطیر دانش برای انتقال دانش از مدل بزرگتر و پیچیده تر (شبکه معلم) به مدل کوچکتر و ساده تر (شبکه دانش آموزی) استفاده می کند. فرآیند یادگیری این معماری شامل شبکه معلم است که بر روی یک مجموعه داده بزرگ آموزش داده می شود و دقت بالایی دارد و شبکه دانش آموزی که بر روی همان مجموعه داده اما با پیش بینی های شبکه معلم به عنوان برجسب آموزش داده می شود. این به عنوان تقطیر دانش شناخته می شود. شبکه دانش آموزی از پیش بینی های شبکه معلم یاد می گیرد و سعی می کند آنها را تقلید کند. این به شبکه دانش آموز اجازه می دهد تا به سطح دقیق مشابه با شبکه معلم دست یابد، اما با مدل کوچکتر و ساده تر.

روند یادگیری به شرح زیر است:

۱. شبکه معلم بر روی داده های آموزشی اصلی آموزش دیده است.
۲. soft labels با شبکه معلم تولید شده اند.
۳. شبکه دانش آموز با وزن های تصادفی مقداردهی اولیه می شود.
۴. شبکه دانش آموز برای به حداقل رساندن عملکرد ضرر(loss) زیر آموزش دیده است:

$$\text{loss} = \alpha * \text{student_loss} + \beta * \text{distillation_loss}$$

$$\text{loss} = \alpha * \text{loss_ce}(\text{hard_prediction}, \text{ground_truth}) + \beta * \text{loss_kl}(\text{soft_predictions}, \text{soft_labels})$$

به طوری که:

- loss_ce ضرر آنتروپی متقابل (cross-entropy loss) است.
- loss_kl واگرایی Kullback-Leibler divergence (Kullback-Leibler divergence) است.
- hard_prediction خروجی شبکه دانش آموز با دمای $T = 1$ است.
- soft_predictions خروجی شبکه دانش آموز با دمای $T = t$ است.

- $T = \text{Temperature}$ خروجی شبکه معلم با دمای t است.
- برجسب های واقعی داده ها (hard labels) ground_truth
- α و β های پارامترهایی هستند که به ترتیب وزن ضرر آنتروپی متقطع و واگرایی Kullback-Leibler را کنترل می کنند.

Kullback-Leibler cross-entropy loss تضمین می کند که شبکه دانش آموز یاد می گیرد که پیش بینی های درست را انجام دهد. واگرایی-Kullback-Leibler شبکه دانش آموز را به دلیل داشتن توزیع منفاوت خروجی نسبت به شبکه معلم جرمیه می کند. این شبکه دانش آموزی را تشویق می کند تا بازنمایی زیربنایی داده ها را مانند شبکه معلم بیاموزد.

شبکه دانش آموز تا زمانی که عملکرد ضرر به حداقل برسد آموزش داده می شود. هنگامی که شبکه دانش آموزی آموزش داده شد، می توان آن را بدون نیاز به شبکه معلم در تولید مستقر کرد.

در اینجا توضیحی گام به گام در مورد نحوه به روز رسانی اوزان شبکه دانش آموز آورده شده است:

۱. شبکه دانش آموزی یک دسته از داده های آموزشی تغذیه می شود.
۲. خروجی های شبکه دانش آموزی محاسبه می شود.
- ۳.تابع ضرر با استفاده از خروجی های شبکه دانش آموز، خروجی های شبکه معلم و برجسب های واقعی محاسبه می شود.
۴. گرادیان تابع ضرر با توجه به وزن شبکه دانشجویی محاسبه می شود.
۵. وزن شبکه دانش آموزی با استفاده از گرادیان و بهینه ساز کاهش گرادیان به روز می شود.

این روند تا زمانی که شبکه دانش آموز آموزش ببیند تکرار می شود.

توضیح پیاده سازی تقطیر دانش با جزئیات بیشتر:

پیاده سازی تقطیر دانش [ویرایش]

شبکه های عصبی (به انگلیسی: Neural Network) معمولاً توزیع احتمال دسته ها در لایه آخر را به کمک تابع پیشینه هموار (به انگلیسی: Softmax) محاسبه می کنند. اگر z_i ورودی z ام لایه آخر (ورودی پیشینه هموار) باشد، احتمال q_i را با مقایسه z_i با دیگر ورودی های این لایه، حساب می کند.

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

دققت کنید که T در رابطه بالا دما است که معمولاً برابر ۱ قرار داده می شود. بدیهی است که T اینجا یک فراپارامتر (به انگلیسی: Hyperparameter) است. هر چه قدر T بیشتر باشد، توزیع ما هموارتر می شود و هرچه قدر کمتر باشد اهمیت z_i ها بیشتر می شود و توزیع ناهموارتر می شود. به تعبیر دیگر هر چه قدر که T بیشتر می شود، احتمال دسته محتمل تر به سمت یک میل می کند ولی هر چه قدر که T کمتر می شود، احتمال رده ها به سمت یکنواخت شدن حرکت می کند.^[۱]

در حالت ساده و ابتدایی، تابع هدف را می توان آنتروپی متقطع (به انگلیسی: Cross Entropy) با اهداف نرم در نظر گرفت. در واقع پس از آموزش دادن مدل معلم در دمای بالا، اهداف نرم به دست می آیند و مدل دانش آموز به کمک آن ها در همان دمای بالا آموزش می بیند. البته بعد از تمام شدن آموزش، مدل دانش آموز از دمای ۱ استفاده می کند.

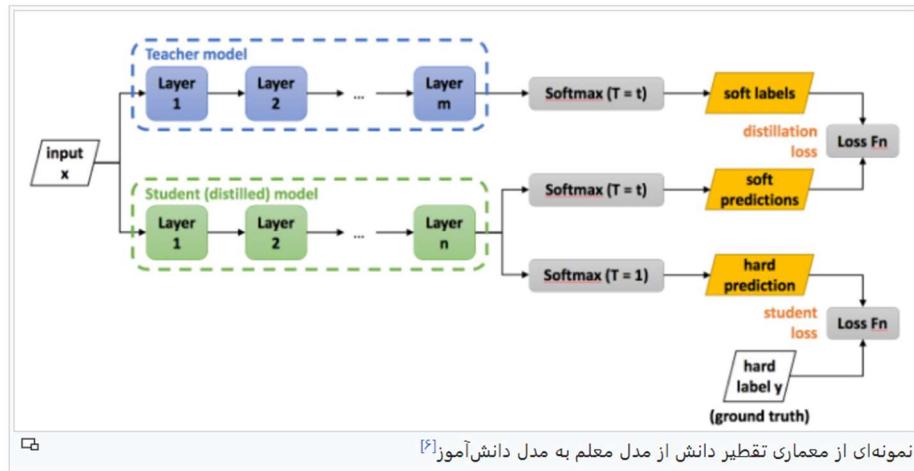
اگر برجسب های درست برای تمام یا بعضی از داده ها در مجموعه انتقال (به انگلیسی: Transfer Set) (مجموعه داده های که از آن برای آموزش مدل دانش آموز استفاده می کنیم) موجود باشد، می توان روش را بسیار بهبود بخشید. یک روش این است که از این برجسب های درست استفاده کرد تا در اهداف نرم اصلاحاتی اعمال کنیم اما روش کارآمدتر این است که تابع هدف را میانگین وزن دار آنتروپی متقطع با اهداف نرم و آنتروپی متقطع با اهداف سخت در نظر بگیریم. دقت شود که در آنتروپی متقطع با اهداف نرم دما در هر دو بالا و یکسان در نظر گرفته می شود در حالی که در آنتروپی متقطع با اهداف سخت، دمای مدل دانش آموز همان طور که در بالا گفتیم برابر ۱ است. به طور تجربی این نتیجه به دست آمده که اگر ضریب جمله دوم به طرز قابل توجهی کم باشد و تأثیر اهداف نرم در تابع هدف ما بیشتر باشد، نتیجه به دست آمده بهتر است.

حال سعی می‌کنیم تا توصیف بالا را به صورت ریاضی بینیم، فرض کنید که D مجموعه داده‌های آموزشی ما است. حال دوتایی‌های ورودی و برجسب به صورت $(x, y) \in D$ می‌باشند. فرض کنید که T مدل معلم با پارامترهای θ_T و S مدل دانش‌آموز با پارامترهای θ_S باشد. حال طبق تقطیر دانش هدف کمینه کردن اتلاف نوشته شده در فرمول زیر است:

$$\sum_{(x,y) \in D} L_{KD}(S(x, \theta_S, \gamma), T(x, \theta_T, \gamma)) + \varphi L_{CE}(\hat{y}_s, y)$$

در فرمول بالا، همان آنتروپی متقطع اهداف سخت است که بر روی برجسب‌های \hat{y}_s پیش‌بینی شده توسط مدل دانش‌آموز و برجسب‌های y واقعی داده‌ها

در دمای ۱ محاسبه می‌گردد. همان اتلاف تقطیر است که در واقع آنتروپی متقطع اهداف نرم می‌باشد که از خروجی‌های لایه پیشینه هموار معلم و دانش‌آموز در دمای γ محاسبه می‌شود؛ بنابراین، $S(x, \theta_S, \gamma)$ خروجی لایه پیشینه هموار دانش‌آموز و $T(x, \theta_T, \gamma)$ خروجی لایه پیشینه هموار معلم را نشان می‌دهند. φ هم فراپارامتری است که میزان تأثیر هر یک از این اتلاف‌ها را در مدل نهایی کنترل می‌کند.^[۱]



:KL divergence فرمول

$$KL(p^g, q) + \sum_{m \in A_k} KL(p^m, q)$$

$$KL(p \parallel q) = \sum_i p_i \log \frac{p_i}{q_i}$$

(ب)

وزن‌های شبکه دانش آموز با توجه به تابع ضرری که برای آموزش شبکه دانش آموز استفاده می‌شود به روز می‌شود. این تابع ضرر معمولاً ترکیب وزنی از ضرر های آنتروپی متقطع و واگرایی Kullback-Leibler است، همانطور که در پاسخ قبلی من توضیح داده شد.

`loss = alpha * student_loss + beta * distillation_loss`

`loss = alpha * loss_ce(hard_prediction, ground_truth) + beta * loss_kl(soft_predictions, soft_labels)`

به طوری که:

- loss_ce ضرر آنتروپی متقابل (cross-entropy loss) است.
- loss_kl واگرایی Kullback-Leibler (Kullback-Leibler divergence) است.
- hard_prediction خروجی شبکه دانش آموز با دمای $T = 1$ است.
- soft_predictions خروجی شبکه دانش آموز با دمای $T = t$ است.

- $T = \text{Temperature}$ با دمای t است.
- soft_labels برچسب های واقعی داده ها (hard labels) هستند.
- α و β های پارامترهایی هستند که به ترتیب وزن ضرر آنتروپی متقاطع و واگرایی Kullback-Leibler را کنترل می کنند.

وزن شبکه دانش آموز با استفاده از بھینه ساز کاهش گرادیان، مانند Adam یا SGD به روز می شود. بھینه ساز کاهش گرادیان با به روز رسانی وزن های شبکه دانش آموزی در جهتی که بیشترین کاهش تابع ضرر را داشته باشد، تابع ضرر را به حداقل می رساند.

در اینجا توضیحی گام به گام در مورد نحوه به روز رسانی اوزان شبکه دانش آموز آورده شده است:

۱. شبکه دانش آموزی یک دسته از داده های آموزشی تغذیه می شود.
۲. خروجی های شبکه دانش آموزی محاسبه می شود.
۳. تابع ضرر با استفاده از خروجی های شبکه دانش آموز، خروجی های شبکه معلم و برچسب های واقعی محاسبه می شود.
۴. گرادیان تابع ضرر با توجه به وزن شبکه دانشجویی محاسبه می شود.
۵. وزن شبکه دانش آموزی با استفاده از گرادیان و بھینه ساز کاهش گرادیان به روز می شود.

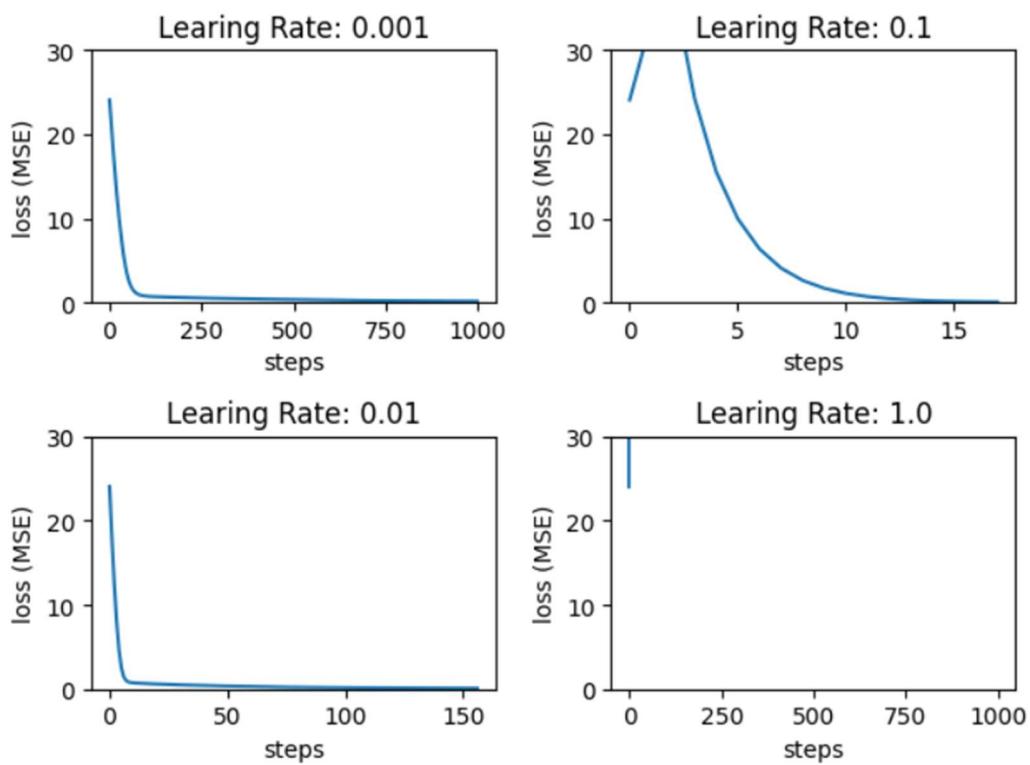
این روند تا زمانی که شبکه دانش آموز آموزش ببیند تکرار می شود.

مراجع:

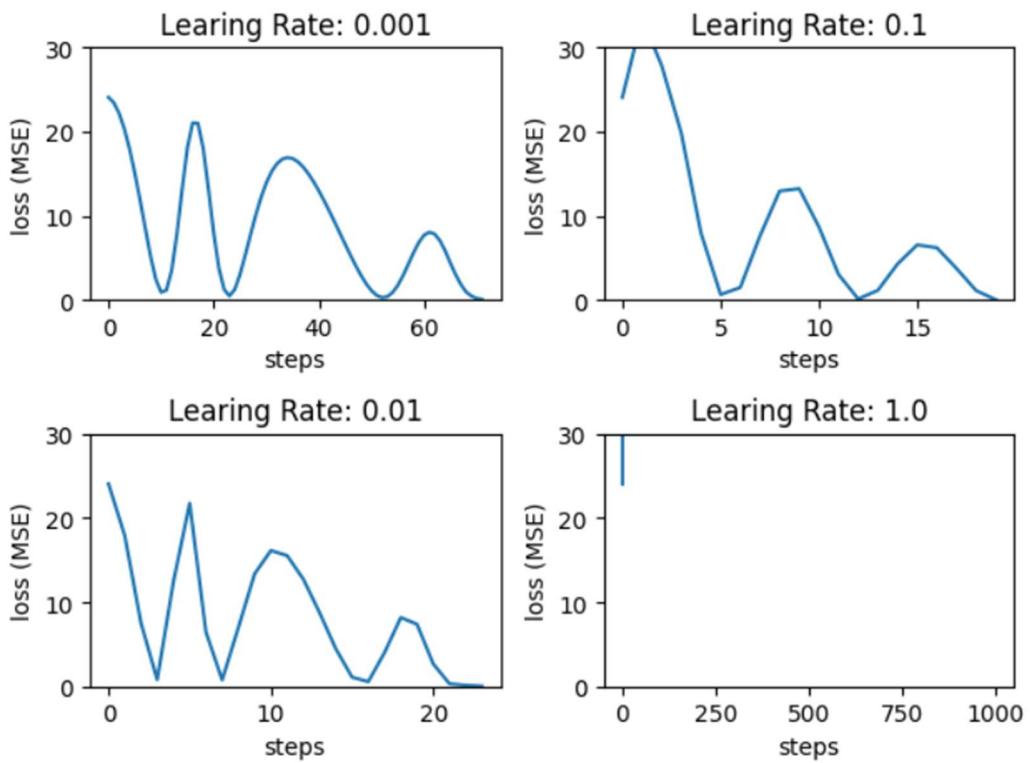
- <https://chat.openai.com/>
- <https://bard.google.com/>
- <https://claude.ai/chats>
- https://fa.wikipedia.org/wiki/%D8%AA%D9%82%D8%B7%DB%8C%D8%B1_%D8%AF%D8%A7%D9%86%D8%B4
- https://intellabs.github.io/distiller/knowledge_distillation.html
- <https://medium.com/neuralmachine/knowledge-distillation-dc241d7c2322>

سوال چهارم

نتایج بدست آمده از بهینه ساز SGD:



نتایج بدست آمده از بهینه ساز momentum:



تجزیه و تحلیل نتایج بدست آمده از بهینه سازهای مختلف:

همان طور که مشاهده می کنید برای نرخ آموزش 1.0 هر دو بهینه ساز عملکرد مشابه و بدی داشته اند. هر دو واگرا شده اند و نقطه مطلوب و خوبی نرسیده اند و بعد از 1000 مرحله فقط بدتر شده اند و ضرر را افزایش داده اند. این به این دلیل است که نرخ آموزش 1.0 بسیار بزرگ است. برای نرخ آموزش 0.1 هر دو بهینه ساز عملکرد عالی و مشابهی داشته اند و در کمتر از ۲۰ مرحله به ضرر تقریباً . رسیده اند. در مابقی نرخ آموزش ها(0.001, 0.01) هر دو مدل به دقت های خوبی رسیده اند(ضرر ها به نزدیکی . رسیده است) ولی momentum زود تر همگرا شده است که نشان دهنده این است که حساسیت کمتری به نرخ آموزش دارد و سریع تر همگرا می شود. Momentum در نرخ آموزش 0.01 حدود ۱۳۰ مرحله زود تر و در نرخ آموزش 0.001 حدود ۹۳۰ مرحله زود تر همگرا شده است. البته این را هم باید در نظر داشته باشیم SGD ساده تر و سریع است و سریار محاسباتی و حافظه ای کمتری دارد.

(کاهش گرادیان تصادفی) SGD

یک بهینه ساز ساده و موثر است که پارامترهای مدل را در جهت گرادیان منفی تابع ضرر به روز می کند. با این حال، SGD می تواند کند همگرا شود و همچنین می تواند به نرخ یادگیری حساس باشد.

در نوت بوک ارائه شده، SGD قادر است مدل را با دقت معقولی آموزش دهد(جز نرخ آموزش 1)، اما برای همگرا شدن به تعداد نسبتاً زیادی تکرار نیاز دارد. علاوه بر این، نرخ یادگیری تأثیر قابل توجهی بر عملکرد SGD دارد. نرخ یادگیری بیش از حد بالا می تواند باعث نوسان مدل و عدم همگرا شود، در حالی که نرخ یادگیری بسیار پایین می تواند باعث شود که مدل به کندی همگرا شود.

Momentum

Momentum نوعی از SGD است که یک جمله تکانه(momentum) را به به روز رسانی گرادیان اضافه می کند. این عبارت momentum به مدل کمک می کند تا در جهت گرادیان منفی تابع ضرر حرکت کند، حتی اگر گرادیان نویزدار یا تعییر جهت باشد. در نوت بوک ارائه شده، momentum قادر است مدل را با دقت بالاتری نسبت به SGD در تکرارهای کمتر آموزش دهد(جز نرخ آموزش 1). علاوه بر این، momentum نسبت به SGD حساسیت کمتری نسبت به نرخ یادگیری دارد. همچنین وجود momentum term می تواند به فرار از نقاط زیپی و همچنین مینیمم های محلی کمک کند.

نتیجه:

به طور کلی، momentum بهینه سازی بهتری نسبت به SGD است، زیرا سریع تر همگرا می شود و حساسیت کمتری نسبت به نرخ یادگیری دارد. با این حال، SGD هنوز هم می تواند انتخاب خوبی برای مسائل ساده باشد یا اگر منابع محاسباتی محدود باشد. تابع SGD به سادگی حاصل ضرب نرخ یادگیری و گرادیان را از پارامتر کم می کند. تابع momentum پک momentum term به روز رسانی گرادیان اضافه می کند. momentum term با در نظر گرفتن میانگین وزنی گرادیان جاری و momentum term قبلی محاسبه می شود.

یادداشت های اضافی:

نرخ یادگیری یک فراپارامتر است که سرعت یادگیری مدل را کنترل می کند. نرخ یادگیری بالاتر می تواند باعث یادگیری سریع مدل شود، اما همچنین می تواند منجر به بی ثباتی شود. نرخ یادگیری پایین تر می تواند باعث کند مدل یادگیری کند، اما می تواند پایدارتر باشد. ضریب momentum یک فراپارامتر است که قدرت momentum term را کنترل می کند. ضریب تکانه بالاتر می تواند به همگرا شریعتر مدل کمک کند، اما همچنین می تواند مدل را نسبت به نویز در گرادیان ها حساس تر کند.

مراجع:

- <https://chat.openai.com/>
- <https://bard.google.com/>
- <https://claude.ai/chats>

سوال پنجم

توضیحات این سوال به طور کامل در کامنت های کد قابل دیدن است در اینجا توضیح مختصری در مورد آن می دهیم.

(الف)

ابتدا با استفاده از تابع Sequential پایتون مدل دلخواه‌مون را می‌سازیم. ساختار آن به شکل زیر است. در ابتدا از لایه flatten برای تبدیل عکس به بردار یک بعدی استفاده کردیم. بعد لایه ورودی و پنهان با تابع فعال سازی relu و در نهایت لایه خروجی را داریم که از تابع فعال سازی softmax برای تبدیل به توزیع احتمالاتی استفاده می‌کند.

```
model = nn.Sequential(  
    nn.Flatten(),  
    nn.Linear(input_size, 256),  
    nn.ReLU(),  
    nn.Linear(256, 128),  
    nn.ReLU(),  
    nn.Linear(128, out_size),  
    nn.LogSoftmax(dim=1)  
)
```

همچنین با توجه به خواسته سوال تابع ضرر crossentropy و بھینه ساز SGD را پیاده سازی نمودیم.

```
criterion = nn.CrossEntropyLoss()  
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

همان طور که در عکس زیر مشاهده می شود قسمت لوپ آموزش را هم تکمیل کردیم. نتایج اجرا هم میبینید که ضرر به مرور کاهش یافته است.

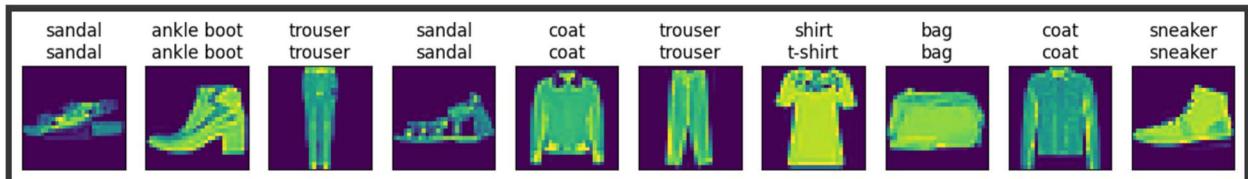
```
# forward pass  
##### Your code #####  
# output = ...  
# loss = ...  
output = model(images)  
# Calculate the loss  
loss = criterion(output, labels)  
#####  
  
# Backward pass to calculate the weight gradients  
loss.backward()  
  
# Update the model parameters  
optimizer.step()  
  
running_loss = running_loss+loss.item()  
else:  
    print(f"Training loss: {running_loss/len(trainloader)}") # print the average loss for the entire epoch
```



```
Training loss: 0.8385571663313584  
Training loss: 0.5013908199918296  
Training loss: 0.44879426610177514  
Training loss: 0.41604458842513914  
Training loss: 0.39337948754207414  
Training loss: 0.37546845778092136  
Training loss: 0.3619414933486534  
Training loss: 0.3493888401337016  
Training loss: 0.3385255498958549  
Training loss: 0.32884212963775533
```

پس از آموزش نیز رو ۰۱ نمونه مدل را تست کردیم که زیر مشاهده میکنید. مدل ما ۹ نمونه را درست پیش بینی کرده است و فقط نتوانسته تفاوت shirt و T-shirt را تشخیص دهد که منطقی هست چون شبیه هم هستند. فقط در این سوال به مشکل ورژن خوردم که با دستور زیر درست شد. همچنین بدون استفاده از flatten در لایه اول مدل هم مشکل ابعاد داشتم.

```
!pip install d2l==0.17.0
```



:مراجع

<https://chat.openai.com/>

<https://bard.google.com/>

<https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html>

سوال ششم

توضیحات این سوال به طور کامل در کامنت های کد قابل دیدن است در اینجا توضیح مختصری در مورد آن می دهیم.(کد این سوال در فایل HW2_Q5.ipynb پیوست شده است).

(ب)

برای بیش برآذش شبکه، می توانیم تغییرات زیر را در مدل تعریف شده در قسمت الف ایجاد کنیم:

- تعداد لایه ها و/یا تعداد نورون ها در هر لایه را افزایش دهیم. این کار به شبکه قدرت، پیچیدگی و پارامترهای بیشتری برای یادگیری می دهد، که می تواند احتمال بیش برآذش داده های آموزشی را افزایش دهد و باعث شود الگوهای نا مربوط و اشتباه توسط مدل حفظ شود. ما در اینجا یک لایه اضافه کردیم و تعداد نورون ها را نسبت به حالت الف بیشتر نمودیم.
- ساختار مدل در زیر آمده است.
- میزان منظم سازی(regularization) را کاهش دهیم. تکنیک های منظم سازی، مانند Dropout و weight decay، با جریمه کردن شبکه به دلیل داشتن وزن های پیچیده، به جلوگیری از برآذش بیش از حد شبکه کمک می کنند. با کاهش میزان منظم سازی، می توانیم احتمال بیش برآذش شبکه را افزایش دهیم. ولی چون در قسمت الف از این تکنیک ها استفاده نکرده بودیم این تغییر را نمیتوانیم اعمال کنیم.
- شبکه را برای مدت زمان طولانی تری آموزش دهید. هر چه شبکه بیشتر آموزش داده شود، احتمال بیشتری وجود دارد که داده های آموزشی بیش برآذش شوند چرا که زمان بیشتری برای حفظ کردن داده های آموزشی دارد. به همین دلیل بجای ۱۰ دوره مدل را ۳۰ دوره آموزش می دهیم.

```
overfit_model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_size, 512),
    nn.ReLU(),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Linear(128, out_size),
    nn.LogSoftmax(dim=1)
)
```

```

epochs_overfit = 30
train_losses = []
valid_losses = []
train_accuracies = []
valid_accuracies = []

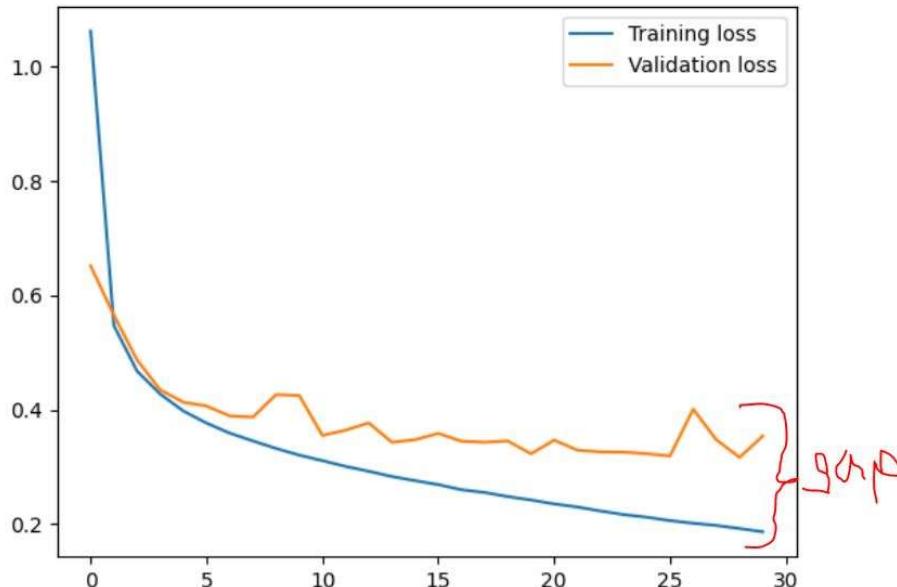
for e in range(epochs_overfit):
    overfit_model.train() # set the model to training mode
    running_loss = 0
    train_sum_accuracy = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer_overfit.zero_grad()
        output = overfit_model(images)
        loss = criterion_overfit(output, labels)
        loss.backward()
        optimizer_overfit.step()
        running_loss = running_loss + loss.item()
        train_sum_accuracy += (output.argmax(1) == labels).type(torch.float).sum().item()
    train_accuracy = train_sum_accuracy / len(trainloader.dataset)

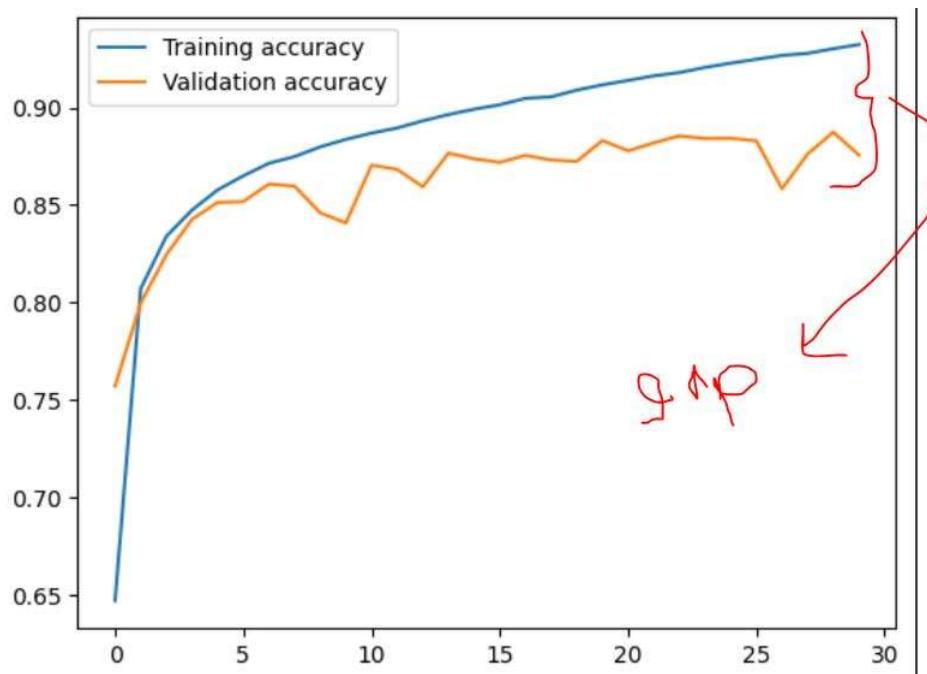
    overfit_model.eval() # set the model to evaluation mode
    test_loss = 0
    test_sum_accuracy = 0

    # Turn off gradients for validation, saves memory and computations
    with torch.no_grad():
        for images, labels in testloader:
            images = images.view(images.shape[0], -1)

```

نمودارهای دقت و ضرر در زیر نشان می‌دهند که فاصله‌ای بین دقت و ضررهای آموزش و تست وجود دارد و عملکرد آموزش از تست بهتر است. پس مدل بیش برآن شده است روی داده‌های آموزشی. چراکه این مدل دارای لایه‌ها و نورون‌های بیشتری در هر لایه نسبت به مدل در قسمت الف است و برای مدت زمان طولانی تری آموزش داده می‌شود.





(پ)

داده‌افزایی (Augmentation)

- بهترین راه برای افزایش قدرت تعمیم‌دهی یک الگوریتم یادگیری ماشین با ظرفیت یادگیری بالا، آموزش آن بر روی داده‌های بیشتر است
- جمع‌آوری داده معمولاً فرآیند دشوار و خسته‌کننده‌ای است
- می‌توانیم داده‌های ساختگی بسازیم و به داده‌های آموزشی اضافه کنیم

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

- این کار برای مسئله دسته‌بندی راحت‌ترین است
- می‌توانیم جفت‌های (x, y) جدید را به سادگی و تنها با تبدیل x بسازیم



برای بهبود شبکه بیش برآذش شده با داده افزایی، می توانیم تبدیل های مختلفی مانند RandomHorizontalFlip و غیره را در مجموعه داده های آموزشی اعمال کنیم. این تغییرات به ایجاد تغییراتی از داده های آموزشی اصلی کمک می کند، از حفظ نمونه های آموزشی دقیق توسط مدل و ترویج تعمیم بهتر جلوگیری می کند.

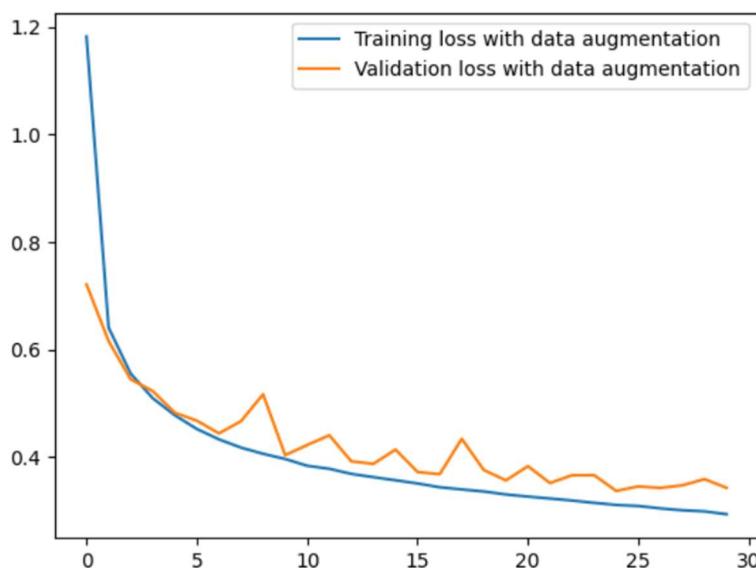
می توانیم از ماثول torchvision.transforms برای اعمال این تبدیل ها در مجموعه داده آموزشی استفاده کنیم. پس از اعمال افزایش داده ها، می توانیم مدل را دوباره آموزش دهیم و مشاهده کنیم که آیا مشکل بیش برآذش بهبود می یابد یا خیر.

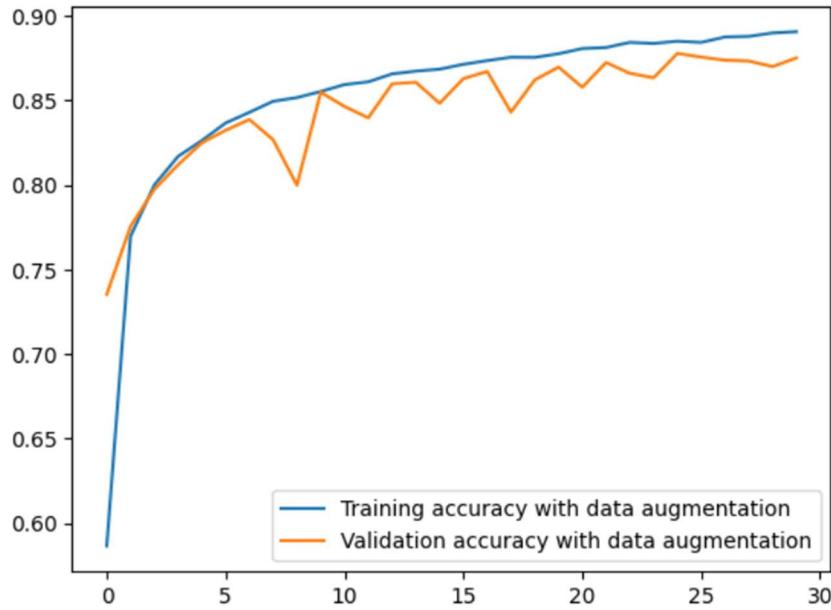
در کد زیر میبینیم که تفاوت آن با مدل قسمت ب که بیش برآذش شده بود فقط در همین قسمت هست که هنگام لود کردن داده ها از داده افزایی استفاده کردیم. مدل همان مدل قسمت ب می باشد.

```
# Data augmentation
transform_augmented = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    # transforms.RandomCrop(size=28, padding=4),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,)))
])

trainset_augmented = datasets.FashionMNIST('./data', download=True,
train=True, transform=transform_augmented)
trainloader_augmented = torch.utils.data.DataLoader(trainset_augmented,
batch_size=64, shuffle=True)
```

ما در اینجا از دو نوع تبدیل برای داده افزایی استفاده کردیم ولی می توانیم از تبدیلات بیشتر مانند RandomCrop که کامنت شده است بهره بگیریم. بقیه مراحل مانند قسمت قبل می باشد. پس بباید نمودار های بدست آمده را تحلیل کنیم.





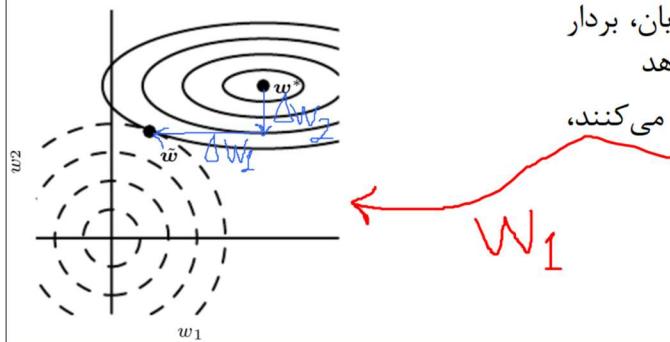
همان طور که میبینید نسبت به قسمت الف با استفاده از داده افزایی تعمیم دهی مدل بهتر شده است. در واقع فاصله (gap) بین دقیق و ضرر آموزش و تست کاهش یافته است (دقیق آموزش کمتر شده، دقیق تست بیشتر شده، ضرر آموزش بیشتر شده و ضرر تست کمتر شده). این به این معنی است که مدل الگوی های مناسب و درستی را یاد گرفته است و بجای حفظ کردن داده آموزشی، تعمیم دهی میکند. پس داده افزایی می تواند به بهبود مشکل بیش برآش کمک کند.

(ت)

منظم سازی پارامتر L2

$$\Omega(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w} - \eta\nabla_{\mathbf{w}}L(\mathbf{w}, b)$$



- قبل از انجام بهروزرسانی معمولی مبتنی بر گرادیان، بردار وزن را در هر گام با یک ضریب ثابت کاهش می دهد

- وزن هایی که تغییر کمتری در تابع ضرر ایجاد می کنند، اهمیت کمتری دارند و بیشتر کاهش می یابند

منظمسازی پارامتر L1

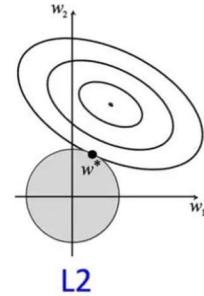
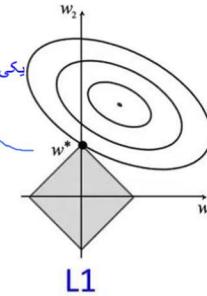
- منظمسازی L1 بر روی پارامترهای شبکه \mathbf{w} به صورت زیر تعریف می‌شود

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_i |w_i|$$

$$\tilde{L}(\mathbf{w}, b) = L(\mathbf{w}, b) + \lambda \|\mathbf{w}\|_1$$

$$\nabla_{\mathbf{w}} \tilde{L}(\mathbf{w}, b) = \nabla_{\mathbf{w}} L(\mathbf{w}, b) + \lambda \text{sign}(\mathbf{w})$$

بک از پارامتر ها را صفر میکنند معمولاً



برای بهبود شبکه با استفاده از تنظیمسازی L1 یا L2، می‌توانیم یک عبارت منظمسازی به تابع ضرر اضافه کنیم. عبارت منظمسازی مدل را به دلیل داشتن وزن‌های بزرگ جریمه می‌کند، که می‌تواند به جلوگیری از برآذش بیش از حد کمک کند. ما هر دو را پیاده سازی و تست کردیم.

منظمسازی L1:

منظمسازی L1 مقدار مطلق وزن‌ها را به تابع کاهش اضافه می‌کند. این می‌تواند به پراکندگی مدل کمک کند، به این معنی که وزن‌های غیر صفر کمتری خواهد داشت. این می‌تواند برای جلوگیری از برآذش بیش از حد مفید باشد، زیرا می‌تواند مدل را مجبور به یادگیری ویژگی‌های مهم تر کند.

برای پیاده سازی مشابه قسمت ب عمل میکنیم با این تفاوت که فقط به تابع ضرر عبارت L1 را اضافه می‌کنیم. کد آن در شکل زیر نشان داده شده است.

```

lambda_l1 = 0.001 # L1 regularization parameter

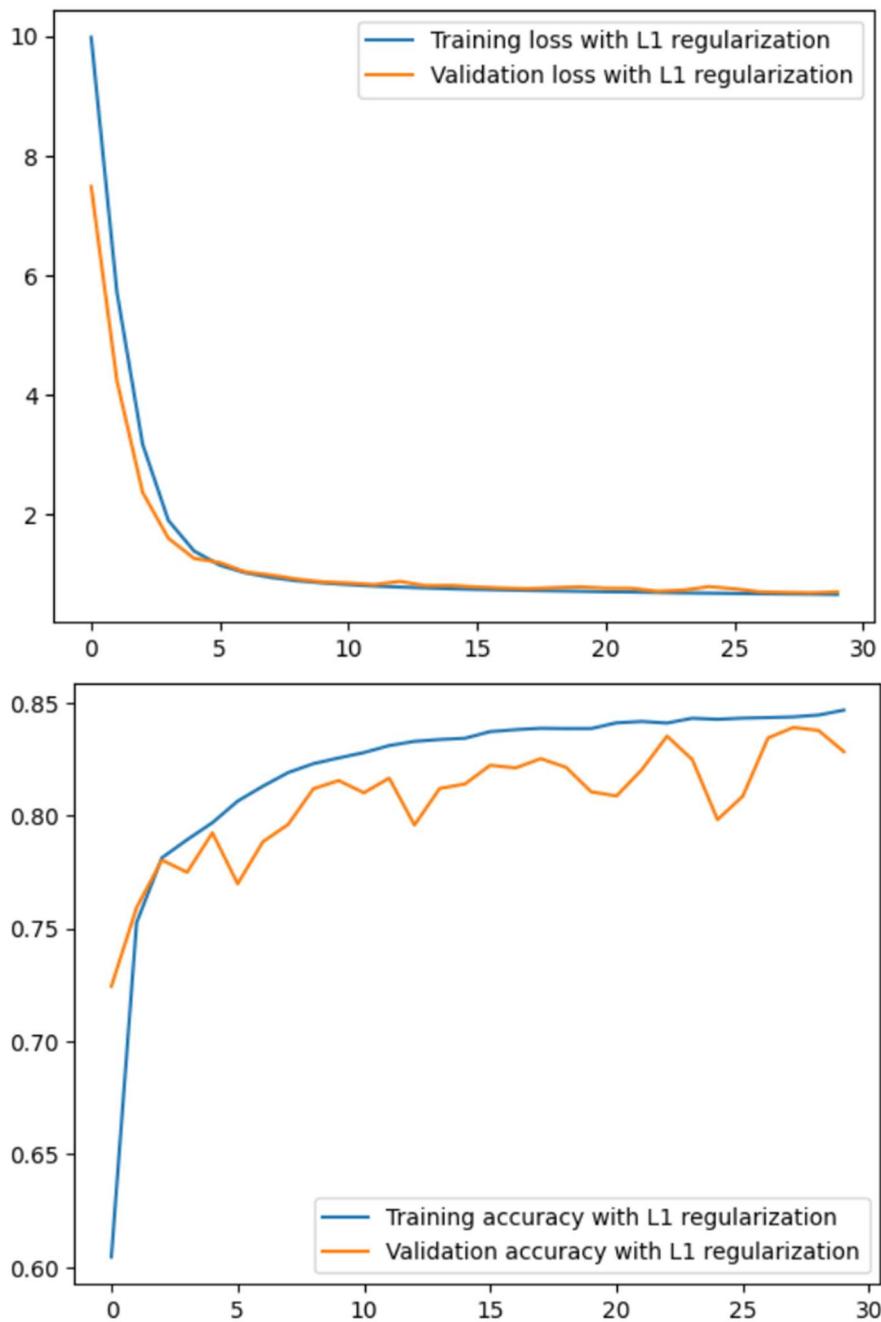
# Train the model with L1 regularization
epochs_l1 = 30
train_losses = []
valid_losses = []
train_accuracies = []
valid_accuracies = []

for e in range(epochs_l1):
    l1_model.train() # set the model to training mode
    running_loss = 0
    train_running_accuracy = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer_l1.zero_grad()
        output = l1_model(images)
        loss = criterion_l1(output, labels)
        l1_penalty = 0
        for param in l1_model.parameters():
            l1_penalty += torch.norm(param, 1)
        loss += lambda_l1 * l1_penalty # Add L1 regularization term to the loss
        loss.backward()
        optimizer_l1.step()
        running_loss = running_loss + loss.item()
        train_running_accuracy += (output.argmax(1) == labels.type(torch.float)).sum().item()
    train_accuracy = train_running_accuracy / len(trainloader.dataset)

    l1_model.eval() # set the model to evaluation mode

```

همچنین در زیر نمودار های نتایج آن را می بینیم.



همان طور که میبینید نسبت به قسمت الف با استفاده از منظم سازی L1 تعمیم دهنده مدل بهتر شده است. در واقع فاصله(gap) بین دقت و ضرر آموزش و تست کاهش یافته است(دقت آموزش کمتر شده، دقت تست بیشتر شده، ضرر آموزش بیشتر شده و ضرر تست کمتر شده). این به این معنی است که مدل الگوی های مناسب و درستی را یاد گرفته است و بجای حفظ کردن داده آموزشی، تعمیم دهنده میکند. پس داده افزایی می تواند به بهبود مشکل بیش برآش کمک کند. ولی عملکرد داده افزایی و منظم سازی L2 نسبت به منظم سازی L1 بهتر است(دقت تست بالاتر است).

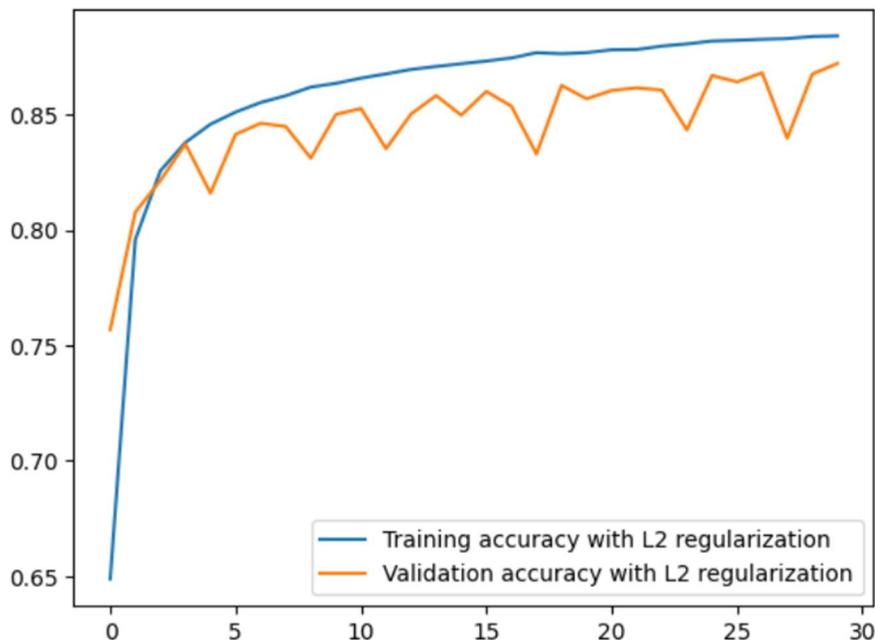
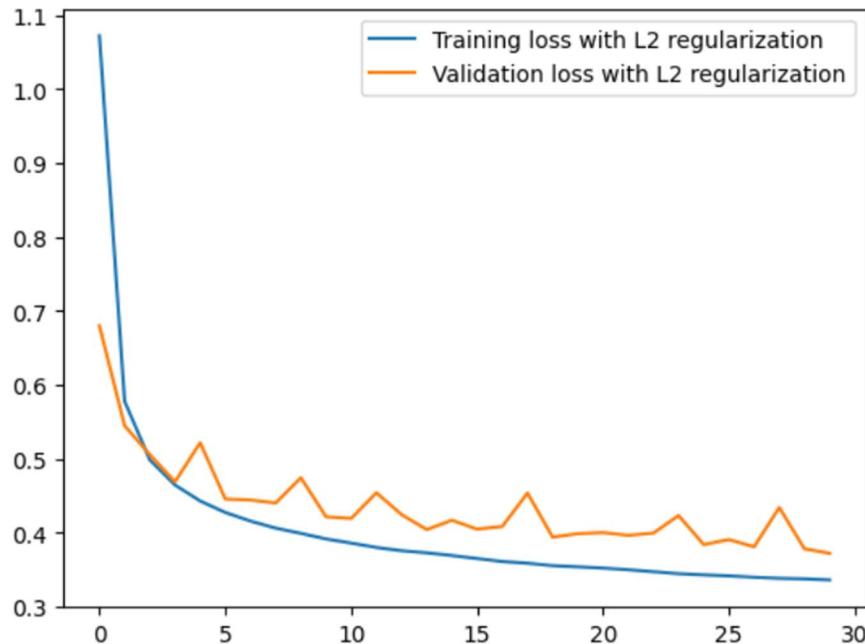
منظمه سازی L2:

منظمه سازی L2 مجدد وزن ها را به تابع کاهش اضافه می کند. این می تواند به صافتر (smoother) شدن مدل کمک کند، به این معنی که احتمال زیاد شدن آن کمتر خواهد بود.

برای پیاده سازی مشابه قسمت ب عمل میکنیم با این تفاوت که فقط به تابع ضرر عبارت L2 را اضافه می کنیم. کد آن در شکل زیر نشان داده شده است. در PyTorch می توانیم از weight_decay در تابع ضرر استفاده کنیم که همان کار را انجام می دهد.

```
# Define the criterion with L2 regularization
criterion_L2 = nn.CrossEntropyLoss()
optimizer_L2 = optim.SGD(l2_model.parameters(), lr=0.01, weight_decay=0.01) # weight_decay is the L2 regularization parameter
```

همچنین در زیر نمودار های نتایج آن را می بینیم.



همان طور که میبینید نسبت به قسمت الف با استفاده از منظم سازی L2 تعمیم دهی مدل بهتر شده است. در واقع فاصله(gap) بین دقت و ضرر آموزش و تست کاهش یافته است(دقت آموزش کمتر شده، دقت تست بیشتر شده، ضرر آموزش بیشتر شده و ضرر تست کمتر شده). این به این معنی است که مدل الگوی های مناسب و درستی را یاد گرفته است و بجای حفظ کردن داده آموزشی، تعمیم دهی میکند. پس داده افزایی می تواند به بهبود مشکل بیش برآش کمک کند. عملکرد منظم سازی L2 و داده افزایی تقريبا مشابه هم هست و هر دو نسبت به منظم سازی L1 دقت تست بیشتری دارند.

(ث)

از بين منظم سازی L1 و L2 ، L2 را برو میگزینیم چراکه در بالا نتیجه بهتری داد. حال ترکیب های مختلف منظم سازی L2، Dropout و Data Augmentation استفاده میکنیم و نتایج را مقایسه میکنیم. فقط چون زمان طولانی است هر کدام را در ۱۵ دوره اجرا میکنیم و سپس مقایسه می کنیم.

همچنانیں برای مقایسه عملکرد دقت روی تست(Validation accuracy) را معیار قرار می دهیم چراکه در محاسبه ضرر L2 یک جمله اضافه هم داریم که در اصل برای جرمیه سازی پارامتر هاست نه ضرر واقعی. همچنانیں معیار های آموزش هم منطقی نیست چون مدل آن ها را دیده و روی داده های تست که آن ها را ندیده است باید برآورد را انجام دهیم. اگر در موردی Validation accuracy یکسان یا خیلی نزدیک بهم باشد آنگاه به Training accuracy نگاه میکنیم. در زیر ترکیب های مختلف نتایج دوره ۱۵ ام را مشاهده میکنید. همچنانی نمودارها و نتایج کامل هر ۱۵ دوره در فایل Hw2_Q5 موجود است در اینجا فقط نتیجه مرحله آخر را بررسی میکنیم چون همه آنها خوبی پراکنده می شود.

Dropout, Data Augmentation, L2 regularization

```
Epoch 15/15.. Training loss: 0.491.. Test loss: 0.454.. Training accuracy:
0.824.. Test accuracy: 0.836
```

Dropout, Data Augmentation

```
Epoch 15/15.. Training loss: 0.436.. Test loss: 0.409.. Training accuracy:
0.841.. Test accuracy: 0.852
```

Data Augmentation, L2 regularization

```
Epoch 15/15.. Training loss: 0.443.. Test loss: 0.447.. Training accuracy:
0.843.. Test accuracy: 0.839
```

Dropout, L2 regularization

```
Epoch 15/15.. Training loss: 0.406.. Test loss: 0.408.. Training accuracy:
0.856.. Test accuracy: 0.854
```

با توجه به اعداد بالا و بر اساس Validation accuracy (اگر یکسان بودند) مشاهده میکنیم که بهترین عملکرد ها از بهتر به بدتر به ترتیب زیر است:

- ۱. Dropout, L2 regularization
- ۲. Dropout, Data Augmentation
- ۳. Data Augmentation, L2 regularization
- ۴. Dropout, Data Augmentation, L2 regularization

لذا بهبود حداقلی با Dropout, L2 regularization حاصل می شود. بعد از آن نیز بیشترین بهبود با Data Augmentation حاصل شده است. البته باید در نظر داشته باشیم که این یک مسئله و مدل خاص می باشد و نمی توان گفت که همیشه چنین اتفاقی می افتد.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

لینک چت من با :chat gpt

<https://chat.openai.com/share/b59cb75f-66c2-4bf6-bb7b-a70b1031938b>

پایان