پاییز ۱۳۹۹

دوره‌ی آموزشی یادگیری عمیق با Keras

یاسمن قاسمی

پاییز ۱۳۹۹

یادگیری عمیق با Keras

فصل سوم

دوره‌ی آموزشی یادگیری عمیق با Keras

مقدمه

صفحه ۸۴

مقدمه

در این فصل با اساس شبکه‌های عمیق آشنا می‌شویم. ابتدا مدل‌های ساده و تک لایه و پیاده‌سازی آن‌ها توضیح داده می‌شوند. سپس با چندین مثال کاربردی، مدل‌های پیچیده‌تر و چند لایه را بررسی خواهیم کرد.

مفاهیم پایه‌ای که در این فصل بررسی خواهند شد:

- Forward propagation
- Backpropagation
- Activation Function
- Loss Function و محاسبه‌ی loss
- الگوریتم gradient descent
- ارزیابی مدل
- رخداد overfitting و underfitting

مروری بر فصل‌های قبل

انواع داده و موجودیت‌های شبکه عصبی در keras:

- **Scaler:** به اعداد گفته می‌شود. در حقیقت نوع (type) یک عدد به صورت تکی را scaler می‌نامیم. این نوع از متغیر یک tensor به فرم 0-order است. برای مثال مقادیر دما به صورت scaler بیان می‌شوند.
- **Vector:** به آرایه‌ی (بردار) یک بعدی از اعداد گفته می‌شود. این نوع از متغیر یک tensor به فرم first-order است. سرعت، مثالی از این دسته است زیرا دارای دو مولفه x و y است. هر کدام از این دو مولفه خودشان دارای یک بعد هستند.
- **Matrix:** آرایه‌هایی دو بعدی، که مولفه‌های آن‌ها خود scaler هستند. این نوع از متغیر یک tensor به فرم second-order است. سرعت در طول زمان مثالی از این دسته است.
- **Tensor:** این موجودیت در keras به صورت عمومی استفاده شده و به حالت کلی از موارد بالا tensor گفته می‌شود، اما در برخی از متون tensor بر موجودیت‌های دارای سه بعد (order) یا بالاتر دلالت دارد.

مروری بر فصل‌های قبل

انواع جمع بین موجودیت‌ها در keras:

- **جمع ماتریس با ماتریس:** به صورت درایه به درایه
- **جمع متغیر scaler با ماتریس:** عدد scaler با تمام درایه‌های ماتریس، نظیر به نظیر جمع می‌شود.

Scalar	Vector	Matrix	Tensor
1	$\begin{bmatrix}1\\2\\3\\4\end{bmatrix}$	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} & \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\\\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} & \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\end{bmatrix}$

مروری بر فصل‌های قبل

یادگیری باناظر (supervised learning): نوعی از یادگیری ماشین که در آن هر داده‌ی ورودی (record) به یک خروجی نگاشت می‌شود. هدف از این نوع یادگیری، آموختن نحوه‌ی شکل‌گیری نگاشت انجام شده است. در این روش یادگیری، تمامی داده‌ها در گام آموزش (train) دارای مقداری به عنوان برچسب (label) هستند که معادل نتیجه‌ی خروجی است.

این نوع از یادگیری در مقابل روش‌های بدون ناظر (unsupervised) قرار می‌گیرند.

الگوریتم‌های دسته‌بند (classification algorithms): این الگوریتم‌ها در زیرشاخه‌ی روش‌های باناظر قرار می‌گیرند. هدف استفاده از این الگوریتم‌ها تشخیص کلاس مربوط به هر داده (برچسب) براساس ویژگی‌های آن داده است.

الگوریتم‌های classification در مقابل الگوریتم‌های رگرسیون (regression) گروه‌بندی می‌شوند.

مروری بر فصل‌های قبل

الگوریتم logistic regression: این الگوریتم از خانواده‌ی classification بوده و زیر مجموعه‌ای از روش‌های باناظر است.

در این روش نظیر هر یک از ویژگی‌های واردشونده (feature، بعد)، وزن (w_i) آموخته می‌شود. سپس هر یک از ابعاد ورودی در وزن نظیر خود ضرب شده، و نتیجه‌ی حاصل‌ضرب‌ها با یک‌دیگر و با مقداری عددی به نام bias جمع می‌شوند. خروجی نهایی جمع مذکور، به عنوان ورودی به یک تابع غیر خطی (σ) داده شده و خروجی گره (node، unit، neuron)، ساخته می‌شود.

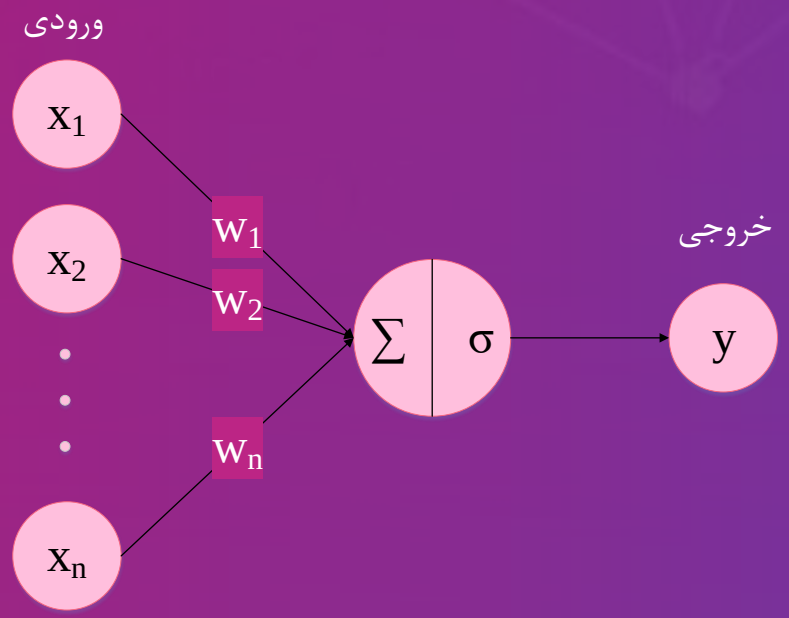
از الگوریتم logistic regression، معمولاً برای پیداکردن احتمال رخداد رویدادهای دو حالتی استفاده می‌شود.

به تابع غیرخطی σ، activation function گفته‌شده و خروجی محاسبات بالا به معنی activation هر گره از لایه‌ی بعد خواهد بود.

مروری بر فصل‌های قبل

در شکل زیر ورودی ما دارای n بعد است و در لایه‌ی وسط، یک نود logistic regression داریم.

$$\sigma(X_1 \times w_1 + X_2 \times w_2 + \cdots + X_n \times w_n + \boldsymbol{b})$$



فایل‌های مورد نیاز در فصل سوم

لینک دانلود دیتاست‌های مورد استفاده در این فصل:

https://github.com/ymgh96/Keras-Programming-Course/tree/main/Chapter%203/data

لینک دانلود کد تمامی تمرین‌ها و فعالیت‌های این فصل:

https://github.com/ymgh96/Keras-Programming-Course/tree/main/Chapter%203

ساخت اولین شبکه عصبی

با تکرار چندین نود logistic regression به صورت پشته‌ای (stack) می‌توان شبکه عصبی را در حالت‌های مختلف پیاده کرد.

لایه ورودی: اولین لایه از شبکه عصبی را لایه‌ی ورودی می‌نامند. در این لایه به تعداد ویژگی‌های تعریف‌شده در دیتاست گره خواهیم داشت. لایه‌ی ورودی معمولا در تصاویر به عنوان سمت چپ‌ترین لایه نمایش داده می‌شود.

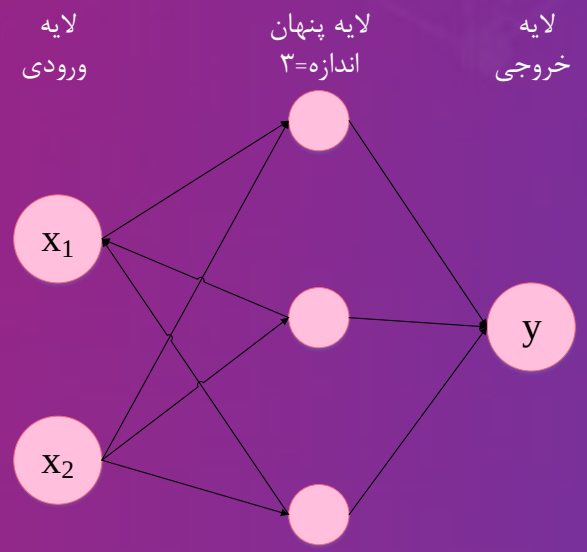
لایه‌ی خروجی: نتیجه و خروجی مدل (label) توسط این لایه ارئه می‌شود. لایه‌ی خروجی معمولا در تصاویر به عنوان سمت راست‌ترین لایه تصویر می‌شود.

شبکه عصبی تک لایه: با به کارگیری چندین نود logistic regression در یک لایه و کنار هم شبکه عصبی تک لایه ساخته می‌شود.

ساخت اولین شبکه عصبی

در شکل زیر یک شبکه‌ی عصبی تک لایه را مشاهده می‌کنید. در این شبکه ورودی ما دارای دو بعد است و هر نود از لایه‌ی ورودی به تمامی نودها از لایه‌ی وسط واردشده و یک hidden layer ساخته شده است.

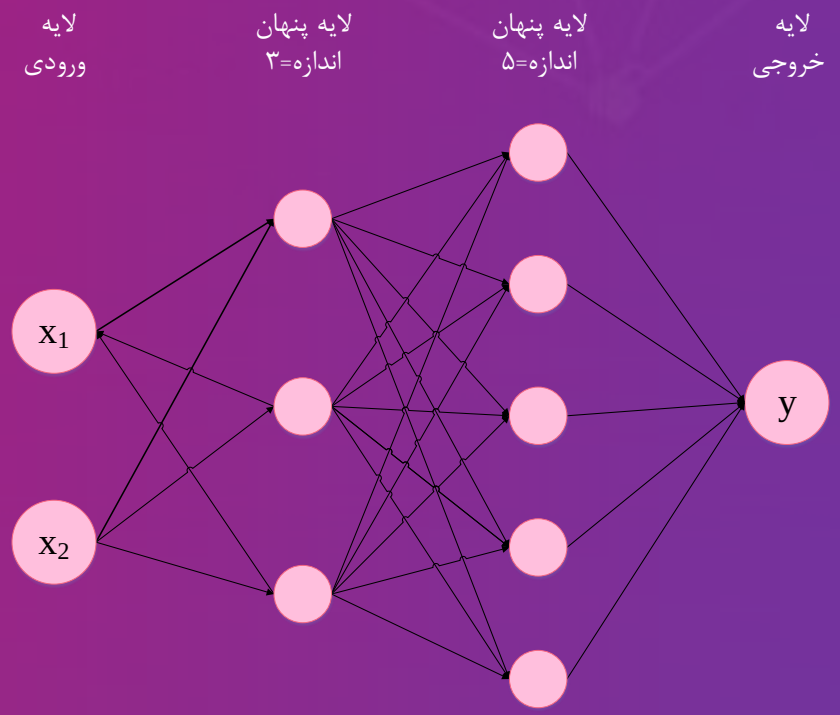
شبکه زیر را fully connected می‌نامیم. زیرا از هر نود، به تمامی نودهای لایه‌ی بعد اتصال وجود دارد.



ساخت اولین شبکه عصبی

شبکه عصبی چند لایه: با پشت سر هم قرار دادن چندین لایه‌ی تکی از شکل قبل، یک شبکه‌ی چند لایه ساخته می‌شود.

شکل زیر یک شبکه‌ی دو لایه است. البته برخی منابع، لایه‌ی ورودی و خروجی را نیز در شمارش درنظر گرفته و این شبکه را چهار لایه می‌دانند.



ساخت اولین شبکه عصبی

پس به صورت کلی هر شبکه یک لایه‌ی ورودی، یک لایه‌ی خروجی و یک یا چند لایه‌ی پنهان دارد.

شبکه‌ی عصبی کم عمق (shallow neural network): شبکه عصبی با یک لایه‌ی پنهان را شبکه عصبی کم عمق می‌نامند.

شبکه‌ی عصبی عمیق (deep neural network): شبکه عصبی با چندین لایه‌ی hidden را شبکه عصبی عمیق می‌نامند.

یادگیری عمیق (deep learning): فرآیند یادگیری شبکه‌های عصبی عمیق را، یادگیری عمیق گوییم.

نکته: هر چقدر شبکه بزرگ‌تر بوده و دارای لایه‌های بیشتری باشد، انعطاف آن بیشتر است. به این معنا که قادر به حل مسائل پیچیده‌تری می‌باشد. اما در مقابل پیچیدگی و تعدد بالای لایه‌ها ملزم به یادگیری وزن‌های بیشتر و هزینه‌ی یادگیری بالاتری خواهد بود.

Hyperparameter: پارامترهایی که توسعه‌دهنده‌ی شبکه عصبی باید در ساخت شبکه به آن‌ها توجه کند. نمونه‌ی این پارامترها عبارت اند از: تعداد لایه‌ها، تعداد گره‌های هر لایه، تعداد epoch ها، تابع خطا (loss function) مورد استفاده و ...

Activation Functions

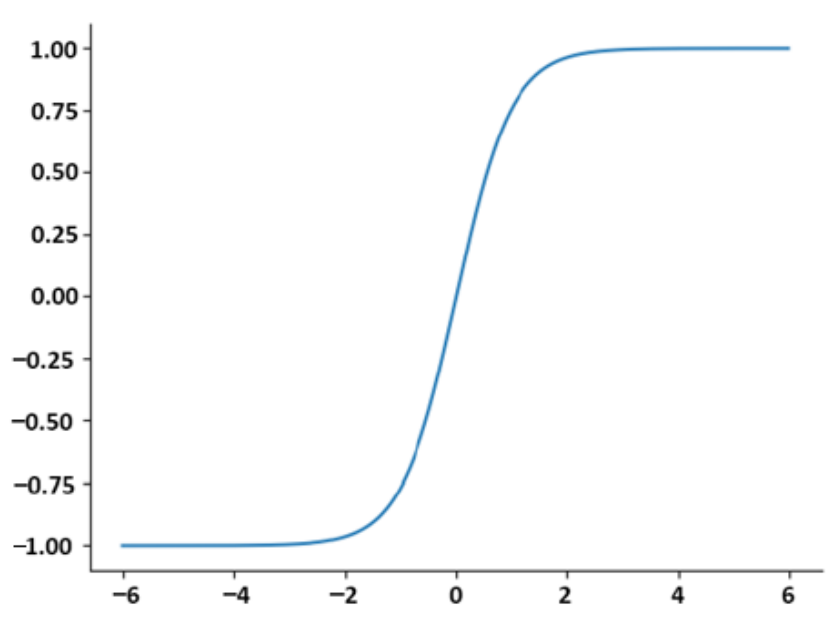
همان‌طور که گفته شد، activation function یک تابع غیر خطی است و به عنوان یکی از گام‌ها، در محاسبات خروجی یک گره مورد استفاده قرار می‌گیرد.

برای هر یک از لایه‌های شبکه عصبی باید یک activation function انتخاب کرد.

انتخاب تابع مورد استفاده به عنوان activation function وابسته به نوع مسئله موردنظر، و تا حدی محل قرارگیری لایه در شبکه است. توابع پر استفاده به عنوان activation function عبارت‌اند از: tanh، ReLU و sigmoid.

Activation Functions

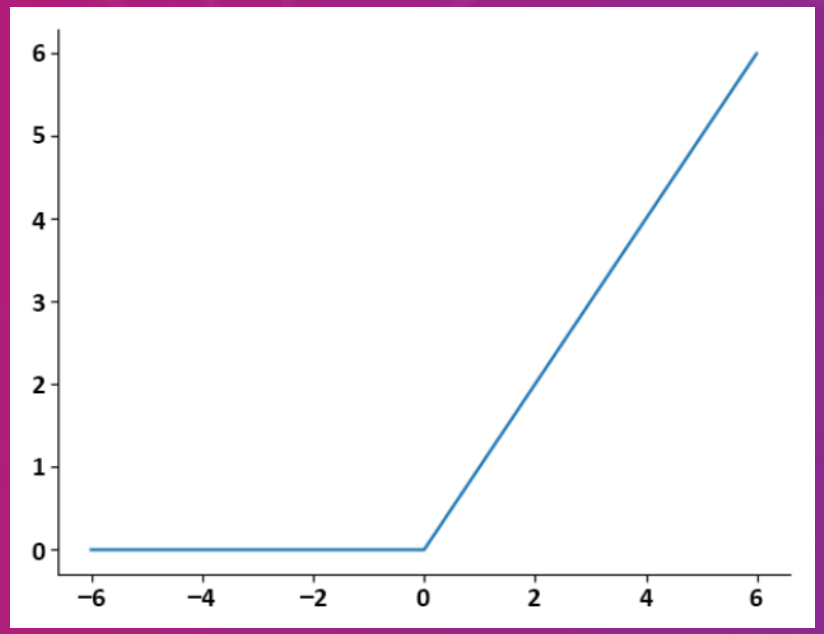
Tanh: از این تابع به عنوان activation function برای لایه‌هایhidden شبکه عصبی استفاده می‌شود این تابع میانگین خروجی‌های هر لایه را نزدیک به صفر نگه می‌دارد.



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

Activation Functions

ReLU: از این تابع نیز برای activation function لایه‌های پنهان استفاده می‌شود. این تابع برای مقادیر کمتر از صفر، شیب صفر و برای مقادیر بیشتر از صفر، شیب ثابت دارد. به همین دلیل بسیار سریع بوده و از پر استفاده‌ترین توابع است.

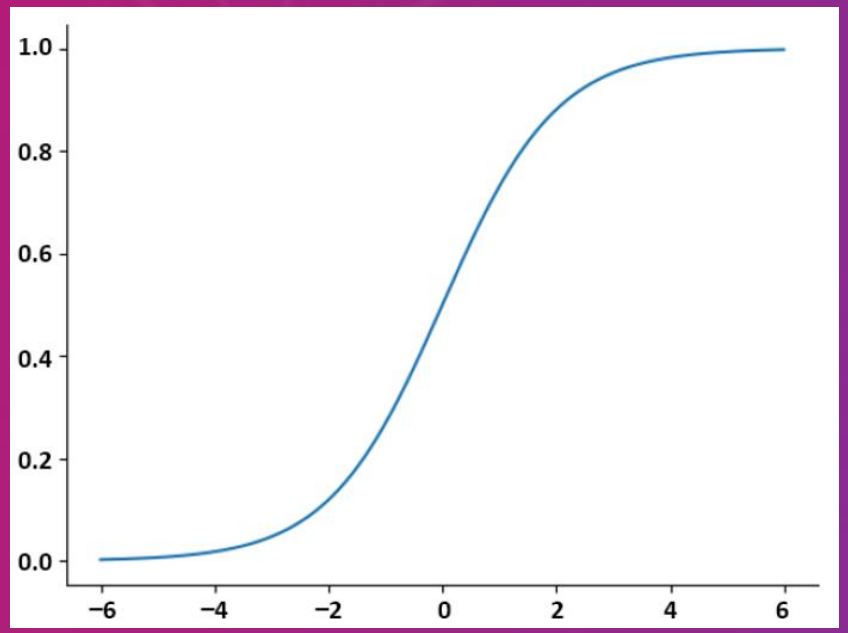


$$\mathrm{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$$\mathrm{ReLU}(x) = \max(0, x)$$

Activation Functions

Sigmoid: در مسائل دو کلاسه (binary classification) از این تابع برای activation function لایه‌ی خروجی استفاده می‌شود. خروجی این تابع مقداری بین صفر تا یک است و به همین دلیل می‌تواند به عنوان احتمال تعلق ورودی به هر یک از دو کلاس تفسیر شود.



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

Forward Propagation

روند شبکه‌های عصبی اصطلاحاً با انتشاری روبه جلو صورت می‌گیرد. روند رو به جلو به این معنی است که ورودی ابتدا از طریق لایه‌ی اول وارد شبکه می‌شود و لایه به لایه عملیات ریاضی انجام شده و خروجی هر لایه به عنوان ورودی لایه بعد در نظر گرفته می‌شود. به این ترتیب خروجی هر لایه در شبکه انتشار می‌یابد.

در شبکه‌ی عصبی بین هر دو لایه، یک ماتریس برای وزن و یک بردار برای مقادیر bias خواهیم داشت.

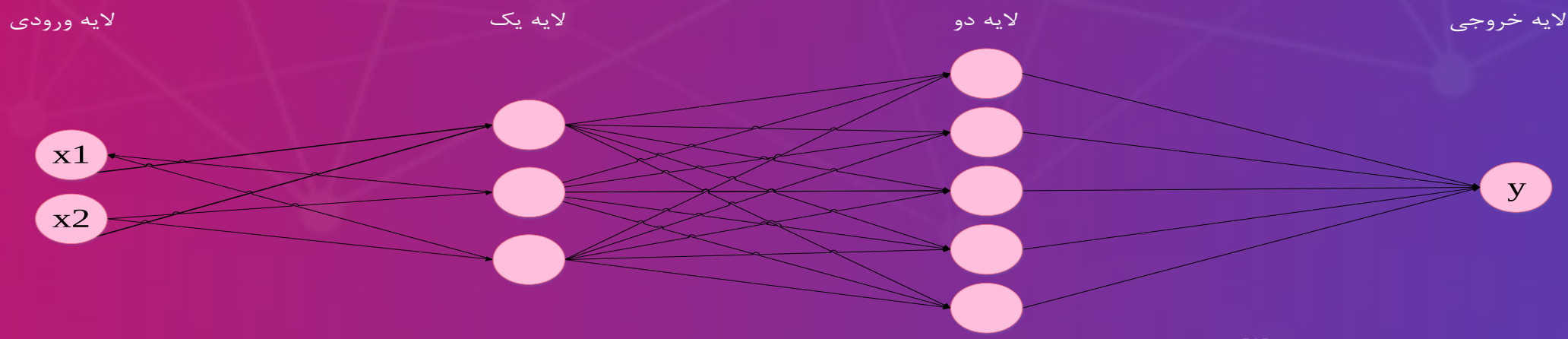
تعداد ماتریس‌های وزن (بردارهای bias) برابر است با:

تعداد لایه‌های hidden + ۱

تعداد کل پارامترهایی که شبکه در فرآیند یادگیری می‌آموزد برابر است با:

تعداد عناصر تمام ماتریس‌های وزن + تعداد عناصر تمام بردارهای bias

Forward Propagation



$$W1 = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix}$$

$$W2 = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} & W_{15} \\ W_{21} & W_{22} & W_{23} & W_{24} & W_{25} \\ W_{31} & W_{32} & W_{33} & W_{34} & W_{35} \end{bmatrix}$$

$$W3 = \begin{bmatrix} W_{11} \\ W_{21} \\ W_{31} \\ W_{41} \\ W_{51} \end{bmatrix}$$

$$b1 = [b_1 \quad b_2 \quad b_3]$$

$$b2 = [b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5]$$

$$b3 = [b_1]$$

مراحل انجام forward propagation

مراحل انجام forward propagation را برای مثال قبل بررسی خواهیم کرد

۱) ورودی شبکه (بردار X) دارای دو ویژگی است، به همین دلیل لایه‌ی ورودی شبکه دارای دو نود می‌باشد. بنابراین ورودی واردشونده به لایه‌ی پنهان اول، از دو تا نود بدست خواهد آمد. برای بدست آوردن مقادیر خروجی لایه‌ی اول (ورودی لایه‌ی دوم یا لایه‌ی پنهان اول) عملیات زیر صورت می‌پذیرند:

$$Z1_{1\times3} = X_{1\times2} \times W1_{2\times3} + b1_{1\times3}$$

$$A1 = tanh(Z1)$$

در معادلات بالا خروجی لایه با نام $A1$ مشخص شده است و اصطلاحاً activation لایه‌ی اول نامیده می‌شود.

همان‌طور که گفته شد activation function در این لایه، تابع tanh است، بنابراین پس از ضرب مقادیر ورودی در وزن‌ها و جمع با bias ($Z1$) باید خروجی به تابع tanh داده شود.

۲) خروجی‌های (activation های) لایه ورودی، به عنوان ورودی به لایه‌ی پنهان اول (لایه‌ی دوم در شکل) داده شده و عملیات زیر برای این لایه نیز تکرار می‌شود:

$$A2 = tanh((A1 \times W2) + b2)$$

مراحل انجام forward propagation

۳) با توجه به اینکه activation function لایه‌ی خروجی تابع sigmoid است، برای این لایه عملیات به صورت زیر انجام می‌شوند:

$$Y = sigmoid((A2 \times W3) + b3)$$

تعداد کل پارامترهای یاد گرفته‌شده در این شبکه برابر است با:

$$|W1| + |b1| + |W2| + |b2| + |W3| + |b3| = 6 + 15 + 5 + 3 + 5 + 1 = 35$$

در آخرین گام از روند یادگیری مدل باید خروجی مدل (Y) را ارزیابی و مقدار loss function بررسی شود.

Loss Function

فرآیند بهینه‌سازی (optimization): در طی یادگیری تا حداقل‌شدن اختلاف خروجی مدل و مقادیر واقعی، مرتباً پارامترها تغییر می‌کنند تا به بهترین حالت (کمترین اختلاف) برسیم.

Loss Function: در حین آموختن پارامترهای شبکه و بدست آوردن بهینه‌ترین پارامترها، باید تابعی تعریف کنیم که خطای بین مقدار واقعی و مقدار اعلام‌شده توسط مدل را اندازه بگیرد، به این تابع loss function گوییم.

در loss function اختلاف خروجی پیش‌بینی شده توسط مدل با مقدار واقعی برچسب برای هر رکورد در دیتاست محاسبه می‌شود.

انواع تعریف loss function:

- در مسائل دسته‌بندی: نسبت داده‌هایی که اشتباه دسته‌بندی شده‌اند.
- در مسائل رگرسیون: میانگین فاصله‌ی بین خروجی واقعی و مقدار پیش‌بینی شده برای تمامی رکوردهای موجود در دیتاست

Loss Function

خلاصه‌ای از loss function های رایج در keras:

- **mean_squared_error**: مناسب مسائل رگرسیون است. به ازای هر نمونه داریم:

$$(\text{خروجی واقعی (برچسب)} - \text{خروجی پیش‌بینی شده})^2$$

- **mean_absolute_error**: مناسب مسائل رگرسیون است. به ازای هر نمونه داریم:

$$|\text{خروجی واقعی (برچسب)} - \text{خروجی پیش‌بینی شده}|$$

- **mean_absolute_percentage_error**: مناسب مسائل رگرسیون است. به ازای هر نمونه داریم:

$$\frac{\text{خروجی واقعی (برچسب)} - \text{خروجی پیش‌بینی شده}}{\text{خروجی واقعی (برچسب)}}$$

Loss Function

خلاصه‌ای از loss function های رایج در keras:

- **binary_crossentropy**: برای دسته‌بندی مسائل دو کلاس استفاده می‌شود. در مواردی که خروجی احتمال و بین صفر و یک است.
- **categorical_crossentropy**: برای classification چند کلاسه مورد استفاده قرار می‌گیرد.

Back Propagation در محاسبه ی مشتق loss function

انتشار روبه عقب (back propagation): فرآیند انجام قاعده‌ی زنجیری (chain rule) در محاسبات از لایه‌ی خروجی به سمت لایه‌ی ورودی را back propagation گویند. به بیان دیگر محاسبه‌ی مشتق loss function در لایه‌ی خروجی و برگشت تاثیر آن به لایه‌های قبلی شبکه را back propagation گوییم.

منظور از مشتق تابع همان مفهوم شیب تابع است. هدف استفاده از مشتق، تعیین جهت تغییر پارامترهای مدل در راستای رسیدن به مقدار کمینه‌ی loss function است.

Chain Rule: loss function تابعی از خروجی پیش‌بینی شده توسط مدل است. از طرفی خروجی مدل، تابعی از وزن‌ها و bias های شبکه است. بنابراین طبق قاعده‌ی زنجیری در مشتق داریم:

$$\frac{d_{loss}}{d_w} = \frac{d_{loss}}{d_{output}} \times \frac{d_{output}}{d_w}$$

گرادیان کاهشی برای یادگیری پارامترها

در این بخش با نحوه‌ی یادگیری پارامترهای بهینه در حین آموزش آشنا خواهیم شد. به بیان دیگر نحوه‌ی به روزرسانی وزن‌ها برای کمینه‌کردن loss function را بررسی خواهیم کرد.

فرآیند یادگیری به صورت تکرارشونده (iterative) صورت می‌گیرد. در هر یک از تکرارها از فرآیند یادگیری، وزن‌ها به نحوی update می‌شوند که به کمینه‌ی loss function نزدیک‌تر شویم. این فرآیند طی الگوریتم‌های بهینه‌سازی (optimization algorithms) انجام می‌شود.

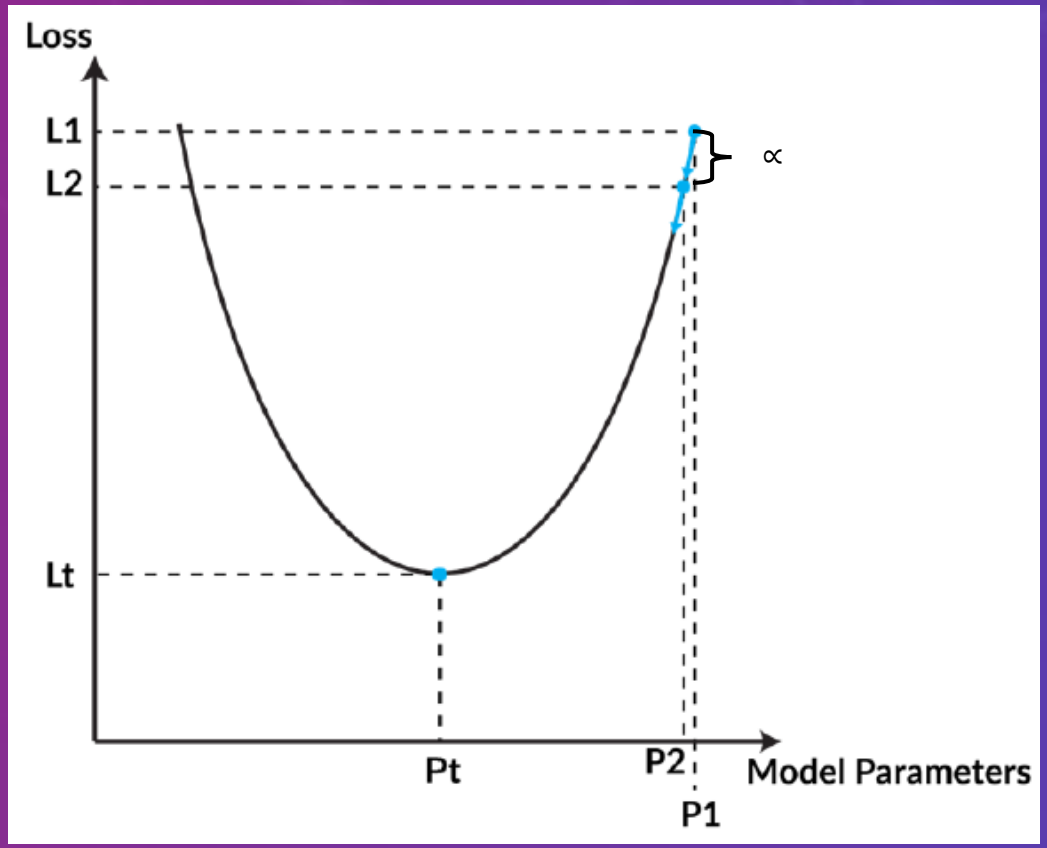
یکی از رایج‌ترین روش‌های بهینه‌سازی استفاده از شیب مشتق تابع است که تحت عنوان الگوریتم بهینه‌سازی گرادیان کاهشی (gradient descent) مورد استفاده قرار می‌گیرد.

گام‌های gradient descent و یادگیری در شبکه عصبی

1) استفاده از forward propagation با پارامترهای فعلی
2) محاسبه‌ی خطا (loss) برای خروجی نظیر تمام دیتاست
3) محاسبه‌ی مشتق loss function نسبت به وزن‌ها و bias دور قبل، در طی back propagation
4) تغییر وزن‌ها در خلاف جهت مشتق و طبق نرخ یادگیری
   - نرخ یادگیری (learning rate, α) مشخص‌کننده‌ی میزان تغییر وزن‌ها نسبت به دور قبل است.
5) تکرار مراحل بالا تا رسیدن به مشتق با مقدار صفر

گام‌های gradient descent و یادگیری در شبکه عصبی

```
Initialize all the weights (w)
and biases (b) arbitrarily
Repeat Until converge {
Compute loss given w and b
Compute derivatives of loss with
respect to w (dw), and with
respect to b (db) using
backpropagation
Update w to w - alpha * dw
Update b to b - alpha * db
}
```



انواع gradient descent

حالت بررسی‌شده از gradient descent، حالت استاندارد از این الگوریتم بود. این روش انواع دیگری نیز دارد.

- **Stochastic Gradient Descent (SGD)**: در این حالت از gradient descent به جای محاسبه‌ی loss و مشتق آن برای تمام رکوردهای دیتاست، در هر تکرار loss را فقط برای زیرمجموعه‌ای از داده‌ها (batch) حساب می‌کنیم.
- **Adam Optimizer**: این روش در حقیقت حالت بهینه‌ی الگوریتم gradient descent است و برای DNN ها اغلب بهتر از SGD عمل می‌کند. در SGD فقط از معیار learning rate برای به روزکردن پارامترها استفاده می‌کردیم، اما در الگوریتم Adam از معیارهایی چون نرخ یادگیری، میانگین وزن‌دار مشتق‌ها و میانگین وزن‌دار مشتق مرتبه دوم نیز در به روزرسانی پارامترها (در هر تکرار) استفاده می‌شود.

برنامه‌نویس در پیاده‌سازی با استفاده از keras، باید بین الگوریتم‌های بهینه‌سازی انتخاب کند.

Batch و Epoch

همان‌طور که گفته‌شد در ساخت شبکه عصبی باید hyperparameter هایی برای فرآیند یادگیری تعیین شوند. در این جا دو تا از این پارامترها را بررسی خواهیم کرد.

اندازه batch (batch size): تعداد داده‌ای که در هر تکرار از الگوریتم بهینه‌سازی به شبکه داده می‌شود را batch گویند.

اگر در keras مقدار پارامتر batch_size را برابر none قرار دهیم، به این معنا است که از batch استفاده نکرده و داده را به مجموعه‌های کوچک‌تر تقسیم نکرده‌ایم. در این‌صورت روش مورد استفاده معادل نسخه‌ی استاندارد gradient descent بوده و در هر بار تکرار الگوریتم کل داده‌های دیتاست به شبکه داده می‌شوند. به بیان دیگر اندازه‌ی batch برابر کل داده‌ها خواهد بود.

Epoch: به تعداد تکرار الگوریتم بهینه‌سازی بر روی تمام داده‌های دیتاست epoch گفته می‌شود. به بیان دیگر تعداد دفعاتی که کل دیتاست به شبکه داده می‌شود را epoch می‌نامند.

مثالی از batch و epoch

مثال: فرض کنید دیتاستی دارای ۴۰۰ رکورد باشد. همچنین اندازه batch_size = 5 و epoch = 20 باشد.

در هر epoch، کل دیتاست (۴۰۰ رکورد) باید در دسته‌هایی شامل ۵ رکورد (batch) تقسیم شود. بنابراین برای اینکه تمام batch ها در یادگیری شرکت کنند،۸۰ تکرار خواهیم داشت.

$$Iteration = \frac{400}{5} = 80$$

طبق رابطه‌ی بالا در هر epoch، ۸۰ تکرار داریم. باتوجه به اینکه برای epoch هم مقدار ۲۰ فرض شده است، تعداد تمام تکرارها در کل epoch ها برابر ۱۶۰۰ تا خواهد بود.

$$Total\ Iterations = 20 \times 80 = 1600$$

نصب کتابخانه‌های پیش‌نیازها

1. نصب پایتون نسخه 3.8
2. به روزکردن نسخه pip به نسخه‌ای بالاتر از 19
3. نصب jupyter notebook
4. نصب tensorflow
5. نصب کتابخانه‌های استفاده‌شده در کدها

```
pip install --upgrade pip

pip install notebook

pip install tensorflow

pip install wheel
pip install numpy
pip install pandas
pip install Keras
pip install matplotlib
```

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

نمونه برنامه‌ای برای پیاده‌سازی یک شبکه عصبی:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/Exercise3_01.ipynb

در این تمرین گام به گام پیاده‌سازی شبکه عصبی را با keras یاد خواهیم گرفت.

دیتاست ما در این مثال، شامل ۱۰ تا ویژگی (ارتفاع، تعداد شاخه، قطر تنه و...) از اطلاعات ۱۰۰۰۰ تا درخت است. اطلاعات رکوردها در فایل tree_class_feats.csv قرار گرفته‌اند.

هدف دسته‌بندی رکوردهای این دیتاست، به دو کلاس برگ ریز و سوزنی برگ است.

برچسب‌ها دارای دو مقدار صفر و یک بوده و در فایل tree_class_target.csv ذخیره شده‌اند. مقدار صفر نظیر درخت سوزنی برگ و مقدار یک به معنی برگ ریز بودن یک درخت است.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

import pandas as pd

کتابخانه‌ی pandas از کتابخانه‌های پایتون برای تحلیل و دستکاری داده‌ها است. برای داده‌های حجیم استفاده از این کتابخانه نسبت به توابع csv ارجحیت دارد.

import numpy as np

کتابخانه‌ی numpy برای انجام عملیات ریاضی بر روی آرایه‌ها و ماتریس‌های بزرگ و چند بعدی کاربرد دارد.

X.shape[0]

X.shape[1]

آرایه‌ی shape از هر متغیری، اطلاعات مربوط به ساختار آن متغیر را نگه می‌دارد. عنصر صفرم این آرایه تعداد سطرهای متغیر و عنصر یکم آن تعداد ستون‌های آن را نگه می‌دارند. تعداد سطرها از X، معادل تعداد رکوردها و تعداد ستون‌های آن معادل تعداد ویژگی‌های دیتاست است.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

np.unique(Y)

تابع unique عناصر منحصر به فرد از متغیر دریافتی را برمی‌گرداند. برای مثال تابع unique در دستور بالا، عناصر منحصر به فرد از لیست برچسب‌ها (تعداد کلاس‌ها) را برمی‌گرداند.

from keras import sequential

Model = Sequential

Model = keras.model.sequential

مدل را sequential انتخاب می‌کنیم، به این معنا که یک stack از لایه‌ها طراحی خواهیم کرد. پس از تعریف یک مدل sequential، می‌توانیم به تعداد دلخواه لایه به مدل اضافه کنیم.

model.add(keras.layers.Dense(10, activation='tanh', input_dim = 10))

با تابع add می‌توانیم به مدل sequential تعریف‌شده یک لایه اضافه کنیم. نوع لایه در این مثال dense و به صورت fully-connected است. پارامتر اول از سازنده‌ی dense مشخص‌کننده‌ی تعداد نورون‌ها در این لایه است. Activation function هر لایه را با مشخص‌کردن ورودی برای لغت کلیدی activation تعیین می‌کنیم. تعداد عناصر واردشونده به لایه نیز با ورودی مربوط به لغت کلیدی input_dim مشخص می‌شوند.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

نکته: پارامتر input_dim فقط برای لایه‌ی اول تعریف می‌شود، زیرا تعداد ویژگی‌های دیتاست (ورودی به لایه اول) برای مدل مشخص نیست. اما در سایر لایه‌ها تعداد یال‌های واردشونده به لایه، با توجه به لایه‌ی قبل و fully connected بودن شبکه مشخص است.

```
model.add(keras.layers.Dense(1, activation='sigmoid')
```

برای لایه‌ی آخر یک نود خواهیم داشت که مقدار صفر یا یک بودن آن متناسب با کلاس تشخیص داده شده توسط مدل خواهد بود.

```
model.compile(optimization='sgd', loss='binary_crossentropy', metrics=['accuracy'])
```

با استفاده از تابع compile در حقیقت پیکربندی (configure) مدل انجام می‌شود. پارامتر optimization از این تابع، مشخص‌کننده‌ی الگوریتم بهینه‌سازی و پارامتر loss مشخص‌کننده‌ی loss function مورد استفاده در شبکه است.

معیار ارزیابی پیش‌فرض در روند آموزش و تست شبکه‌ها loss است. اما می‌توان با پارامتر metrics در قالب یک لیست، معیارهایی دیگری مثل accuracy را نیز اضافه کرده و مقدار آن را در هر خروجی به ازای هر epoch مشاهده کرد.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

```
model.summary()
```

تابع summary خلاصه‌ای از معماری مدل ساخته‌شده را به ازای لایه‌های مختلف ارائه می‌دهد.

```
history = model.fit(X,y,epochs=100,batch_size=5,verbose=1,validation_split=0.2, shuffle=false)
```

انجام train شبکه با فراخوانی و اجرای تابع fit انجام می‌شود. از طریق پارامتر اول از این تابع رکوردها، با پارامتر دوم برچسب‌ها و با لغات کلیدی epoch و batch_size تعداد تکرارهای الگوریتم و اندازه‌ی هر batch را مشخص می‌کنیم.

پارامتر verbose از ورودی‌های تابع fit سه مقدار صفر، یک و یا دو می‌تواند داشته باشد. مقدار صفر هیچ اطلاعاتی از روند train را چاپ نمی‌کند. مقدار یک اطلاعات مربوط به روند آموزش را به صورت کامل چاپ کرده و مقدار دو فقط شماره‌ی epoch ها را چاپ خواهد کرد.

پارامتر validation_split یکی از راه‌های تقسیم دیتاست به زیرمجموعه‌های train و validation است. مقدار این پارامتر عددی بین ۰ تا ۱ خواهد بود و برای مثال 0.2 به این معنا است که ۲۰ درصد آخر از رکوردهای دیتاست برای validation نگه داشته شده و در آموزش استفاده نشوند. به این بخش از داده اصطلاحاً داده‌ی دیده‌نشده (unseen) گفته می‌شود. زیرا مدل در گام آموزش این داده‌ها را در اختیار نداشته و وزن‌های آموخته‌شده مستقل از این داده‌ها می‌باشند.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

پارامتر shuffle می‌تواند یکی از دو مقدار true یا false را داشته باشد. مقدار true برای shuffle به این معنا است که قبل از هر epoch بر کل داده‌های آموزش درهم‌سازی (permutation) رندمی صورت گیرد و سپس batch ها از خروجی درهم‌سازی انتخاب شوند.

```
import matplotlib.pyplot as plt
```

از کتابخانه‌ی matplotlib به منظور ترسیم نمودار و اشکال (visualize) در پایتون استفاده می‌شود.

```
%matplotlib inline
```

برای نمایش نمودارها به صورت تعاملی، از حالت inline استفاده می‌شود. در حالت تعاملی پس از نمایش اشکال، اجرا ادامه می‌یابد. در مقابل در حالت غیر تعاملی با نمایش شکل‌ها اجرا block شده و تا زمانی که پنجره حاوی شکل بسته نشود، اجرا ادامه نمی‌یابد.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین (روش اول رسم نمودار):

plt.plot(history.history[‘accuracy’])

plt.plot(history.history[‘val_accuracy’])

با اجرای دو دستور بالا دو نمودار براساس دقت‌های بدست آمده از آموزش و تست (ذخیره‌شده در history) در یک تصویر (plot) رسم خواهند شد.

plt.title(‘Model Accuracy’)

برای مشخص‌کردن عنوان نمودار از تابع title استفاده می‌کنیم. ورودی این تابع، مقدار مورد نظرمان برای عنوان خواهد بود و به صورت رشته به تابع داده خواهد شد.

plt.ylabel(‘Accuracy’)

plt.xlabel(‘epoch’)

برای مشخص‌کردن عناوین محورهای عمودی و افقی به ترتیب از توابع ylabel و xlabel استفاده می‌کنیم.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

```
plt.legend(['Train','validation'],loc='upper left')
```

برای مشخص‌کردن توصیف نمودارها از تابع legend استفاده می‌شود. یک حالت برای فرخوانی تابع legend مشابه خط بالا است. در این نوع از فراخوانی موارد موجود در پارامتر اول به ترتیب به موارد تعریف‌شده در plot نظیر می‌شوند. در این مثال، لغت "Train" به منحنی اول یعنی accuracy و لغت "validation" به منحنی دوم یعنی val_accuracy نگاشت می‌شود. همچنین با مقدار پارامتر loc می‌توان محل قرارگیری توصیف را در plot تعیین کرد.

```
plt.show()
```

در پایان پس از انجام کامل تنظیمات، با تابع show خروجی نمودارها را نمایش خواهیم داد.

تمرین 3.01: پیاده‌سازی شبکه عصبی با Keras

دستورات مهم در این تمرین:

```
y_predicted = model.predict(X.iloc[0:10,:])
```

برای دسترسی به متغیرها از طریق اندیس، در کتابخانه pandas از iloc استفاده می‌شود. برای نمونه در مثال بالا، تمامی ستون‌ها برای سطرهای ۰ تا ۱۰ از ماتریس X انتخاب خواهند شد.

با تابع predict می‌توان مقدار خروجی پیش‌بینی شده برای داده‌های ورودی (۱۰ تا رکورد اول از دیتاست) را دریافت کرد.

```
np.round(y_predicted)
```

مطابق دستور بالا مقادیر پیش‌بینی شده توسط مدل را با تابع round گرد می‌کنیم. به این ترتیب مقادیر بیش‌تر از ۰.۵ به یک، و مقادیر کمتر از ۰.۵ به صفر نگاشت خواهند شد. به بیان دیگر، اگر مدل برای نمونه‌ای برچسبی بزرگ‌تر از ۰.۵ ارائه دهد آن داده متعلق به کلاس یک و در غیر این‌صورت متعلق به کلاس صفر خواهد بود.

فعالیت 3.01: ساخت شبکه عصبی تک لایه به هدف دسته‌بندی باینری

نمونه برنامه‌ای برای تمرین پیاده‌سازی یک شبکه عصبی:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/Activity3_01.ipynb

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/utils.py

در این فعالیت یک مدل لایه‌ای (sequential) برای انجام binary classification خواهیم ساخت. همچنین با بررسی performance مدل‌ها درک بهتری از نتایج استفاده از تعداد نورون‌های متفاوت خواهیم داشت.

دیتاست مورد استفاده در این مثال مربوط به نتایج تست تولید پروانه‌ی هواپیما است. فرض کنید این دیتاست دارای دو ویژگی است که هر یک نشان‌دهنده‌ی نتیجه‌ی تست‌های دستی انجام‌شده بر پروانه‌ها است. نتایج جمع‌آوری شده در این دیتاست مربوط به ۳۰۰۰ تا پروانه (رکورد) می‌باشد.

هدف دسته‌بندی پروانه‌های طراحی‌شده به دو دسته‌ی قابل قبول (pass) و یا غیرقابل قبول (fail) است. بنابراین مسئله دو کلاسه (صفر به معنای fail و یک به معنای pass) است.

در این فعالیت ابتدا باید یک مدل logistic regression، سپس یک شبکه تک لایه با سه گره و در آخر یک شبکه تک لایه با ۶ تا گره بسازید.

فعالیت را طبق گام‌های تعریف‌شده در کد کامل کنید.

فعالیت 3.01: ساخت شبکه عصبی تک لایه به هدف دسته‌بندی باینری

دستورات مهم در این فعالیت:

```
from sklearn.model_selection import train_test_split
```

با کتابخانه‌ی train_test_split به صورت تصادفی تمام رکوردهای دیتاست برحسب اندازه داده‌شده به زیرمجموعه‌های آموزش و تست تقسیم می‌شوند.

```
import matplotlib.patches as mpatches
```

برای رسم منحنی‌های شخصی‌سازی شده و دلخواه از کتابخانه mpatches استفاده می‌شود.

```
from utils import plot_decision_boundary
```

در فایل utils.py یک تابع به نام plot_decision_boundary نوشته شده است که مرز تصمیم میان دو کلاس را ترسیم خواهد کرد.

فعالیت 3.01: ساخت شبکه عصبی تک لایه به هدف دسته‌بندی باینری

دستورات مهم در این فعالیت:

```
seed = 1

np.random.seed(seed)

tensorflow.random.set_seed(seed)
```

مقادیر پارامترهای اولیه‌ی مدل‌ها به صورت تصادفی انتخاب می‌شوند، بنابراین در صورت نیاز به تکرار یک اجرا لازم است تا مقدار seed اولیه‌ی مورد استفاده توسط numpy و tensorflow ثابت نگه داشته شود. زیرا تمامی مقادیر رندم تولیدشده در این دو کتابخانه براساس مقدار اولیه‌ی seed محاسبه خواهد شد.

```
matplotlib.rcParams['figure.figsize'] = (10.0, 8.0)
```

برای تغییر اندازه پیش‌فرض figure ها از دستور فوق استفاده می‌شود. به صورت کلی تابع rcParams برای شخصی‌سازی خواص ظاهری matplotlib مورد استفاده قرار می‌گیرد.

فعالیت 3.01: ساخت شبکه عصبی تک لایه به هدف دسته‌بندی باینری

دستورات مهم در این فعالیت (روش دوم رسم نمودار):

```
class_1=plt.scatter(feats.loc[target['Class']==0,'feature1'], feats.loc[target['Class']==0,'feature2'], c="red", s=40, edgecolor='k')
```

برای رسم نمودار نقطه‌ای پراکنده از تابع scatter استفاده می‌شود. متغیر اول در فراخوانی این تابع نظیر مختصات x و متغیر دوم نظیر مختصات y نقاط است. استفاده از تابع loc برای دسترسی به اندیس‌های متغیرها به کمک label است. بنابراین در اولین ورودی از دستور بالا ستون با برچسب feature1 برای سطرهایی از ماتریس feats انتخاب می‌شود که مقدار Class در ماتریس target برای آن سطر صفر باشد. به بیان دیگر از دیتاست مقدار feature1 را برای رکوردهای متعلق به کلاس صفر انتخاب می‌کنیم. همچنین مقدار feature2 نیز برای همان رکوردها به عنوان مختصات y درنظر گفته می‌شود. پس نقطه‌ها براساس زوج مرتب [feature1, feature2] برای داده‌های کلاس صفر انتخاب شدند.

پارامترهای c، s و edgecolor نیز به ترتیب مشخص‌کننده‌ی رنگ، اندازه و رنگ لبه‌های نقاط هستند. مقدار k نیز بیان‌کننده‌ی رنگ مشکی (black) است.

فعالیت 3.01: ساخت شبکه عصبی تک لایه به هدف دسته‌بندی باینری

دستورات مهم در این فعالیت:

```
plt.legend((class_1, class_2),('Fail','Pass'))
```

روش دیگر برای فراخوانی تابع legend مشابه بالا است. در این روش لیستی از اشکال تعریف‌شده برای نمایش، به لیستی از برچسب‌ها نظیر می‌شوند. برای نمونه در این مثال، داده‌هایی که با scatter به عنوان class_1 نمایش داده می‌شوند به برچسب Fail نظیر می‌شوند.

انواع لایه‌ها در keras

- **Dense**: ساده‌ترین نوع لایه در keras و لایه‌ای fully connected است. مورد استفاده در مسائل classification و regression شامل داده‌های حجیم
- **Convolutional**: لایه‌ای که موجب ساخت کرنل convolutional می‌شود. مورد استفاده در دسته‌بندی تصاویر
- **Pooling**: این نوع لایه برای کاهش ابعاد لایه‌ی ورودی مناسب است. روش مورد استفاده در الگوریتم‌های pooling به این‌صورت است که برای زیرمجموعه‌ای از داده‌ها با عملیاتی چون میانگین‌گیری و یا پیداکردن حداکثر مقدار، یک مقدار به عنوان نماینده‌ی کل زیرمجموعه تعیین خواهد شد. با این روش بخشی از داده‌ها از دست خواهند رفت، زیرا به جای تمام داده‌های داخل هر زیرمجموعه تنها یک رکورد جایگزین خواهد شد.
- **Recurrent**: لایه‌ای مناسب برای پیداکردن الگوها (patterns) در میان داده‌هایی که ساختار دنباله‌ای (sequence) دارند، مانند داده‌های زبان طبیعی و یا داده‌های time-series

ارزیابی مدل

صفحه ۱۰۵

دوره‌ی آموزشی یادگیری عمیق با Keras

ارزیابی مدل

در این بخش بررسی بیشتری بر ارزیابی مدل‌های چند لایه (deep) انجام می‌دهیم.

تا این قسمت متوجه شدیم که در ساخت هر مدل hyperparameter های زیادی دخالت دارند. همچنین با تعیین مقادیر مختلف برای hyperparameter ها می‌توان برای داده‌های یکسان به نتایج مختلفی رسید.

در این بخش بهترین پارامترها را برای مدل انتخاب کرده و همچنین مباحث بیش‌برازش (overfitting) و کم‌برازش (underfitting) را تشریح خواهیم کرد.

ارزیابی یک مدل Train شده با Keras

روش‌های دیداری و رسم نمودار (visualizing) که پیش از این پیاده‌سازی کردیم، تنها برای مسائل قابل ترسیمی چون مسائل دوبعدی کارایی دارند. در مواردی که تعداد ابعاد مسائل به بیش از سه بعد برسد، امکان ارزیابی مدل از طریق روش visualize ممکن نخواهد بود. بنابراین لازم است تا از روش‌های جامع‌تری برای بررسی کارایی مدل استفاده کرد.

یکی از روش‌های ارزیابی مدل، محاسبه‌ی loss مدل است. به این معنا که مدل برای نمونه‌های مورد پیش‌بینی تا چه حدی دچار خطا شده است. در keras از تابع evaluate به این منظور استفاده خواهیم کرد.

تابع evaluate معمولاً به جای تابع predict استفاده می‌شود.

Predict: خروجی پیش‌بینی مدل برای داده‌ی ورودی

Evaluate: loss مدل در پیش‌بینی کلاس برای داده‌ی ورودی، در مقابل برچسب واقعی آن

ارزیابی یک مدل Train شده با Keras

چندین برنامه برای کار با تابع evaluate و آرگومان‌های آن:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/Evaluating_a_Trained_Model_with_Keras.ipynb

در این مثال با نحوه‌ی کارکرد تابع evaluate آشنا خواهیم شد.

همچنین سایر آرگومان‌های قابل استفاده به عنوان metrics را در keras بررسی می‌کنیم.

دستورات مهم در این مثال:

```
model.evaluate(X, y, batch_size=None, verbose=0)
```

همان‌طور که گفته شد از تابع evaluate برای محاسبه‌ی loss مدل استفاده می‌شود. پارامتر اول این تابع نظیر داده‌ها و پارامتر دوم نظیر برچسب‌های واقعی مدل است. همچنین اندازه‌ی batch مشخص‌کننده‌ی این است که loss روی چه بخشی از دیتاست محاسبه شود، حالت none به معنی محاسبه‌ی loss روی کل دیتاست است.

تقسیم داده به زیرمجموعه‌های آموزش و تست

پارامترهای مدل براساس داده‌های آموزش یاد گرفته می‌شوند. بنابراین برای گام تست باید داده‌ای درنظر گرفت که در گام آموزش شرکت نکرده باشد. به این نوع داده، داده‌ی دیده‌نشده (unseen) گویند.

برای ساخت داده‌ی unseen، کل مجموعه داده را به دو زیر مجموعه‌ی آموزش (training subset) و زیر مجموعه‌ی تست (test subset) تقسیم می‌کنیم.

Training Set: مجموعه‌ای شامل داده‌های مورد استفاده در گام آموزش است. از این مجموعه به هدف فراهم‌آوردن داده‌ی کافی برای یادگیری دقیق روابط و الگوهای موجود در داده استفاده می‌شود.

Test Set: مجموعه‌ای شامل داده‌های مورد استفاده در گام تست است. از این مجموعه به هدف فراهم‌آوردن داده‌ی دیده‌نشده برای مدل و تخمین‌های bias نشده استفاده می‌شود.

تقسیم داده به زیرمجموعه‌های آموزش و تست

نسبت متداول برای تقسیم‌بندی داده‌ها در استفاده از دیتاست‌های کوچک به صورت زیر است:

۷۰ درصد گام آموزش- ۳۰ درصد گام تست

۸۰ درصد گام آموزش- ۲۰ درصد گام تست

نسبت متداول برای تقسیم‌بندی داده‌ها در استفاده از دیتاست‌های حجیم به صورت زیر است:

۹۸ درصد گام آموزش- ۲ درصد گام تست

۹۹ درصد گام آموزش- ۱ درصد گام تست

نکته: مطابق شکل زیر و باتوجه به توضیحات داده‌شده پیرامون داده‌ی unseen، میان دو زیرمجموعه‌ی آموزش و تست هم‌پوشانی وجود نخواهد داشت.

تقسیم داده به زیرمجموعه‌های آموزش و تست

برای تقسیم داده به زیرمجموعه‌های آموزش و تست دو راه وجود دارد:

- استفاده از تابع train_test_split
- استفاده از پارامتر validation_split در تابع fit

نمونه برنامه‌ای برای کار با تابع train_test_split و پارامتر validation_split:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/Splitting_Data_into_Training_and_Test_Sets.ipynb

دستورات مهم در این مثال:

```
from sklearn.model_selection import train_test_split
```

تابع train_test_split در کتابخانه‌ی model_selection از sklearn قرار دارد.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=None)
```

پارامتر اول از تابع train_test_split برابر کل دیتاست و پارامتر دوم برابر برچسب نظیر تمام رکوردها می‌باشد. پارامتر test_size از این تابع مشخص‌کننده‌ی اندازه‌ی زیرمجموعه‌ی تست است و باقی داده‌ها از کل دیتاست به عنوان زیرمجموعه‌ی آموزش درنظر گرفته می‌شوند. یعنی در این مثال ۳۰ درصد از کل رکوردهای موجود در دیتاست برای ارزیابی و ۷۰ درصد آن‌ها برای آموزش درنظر گرفته می‌شوند.

تقسیم داده به زیرمجموعه‌های آموزش و تست

دستورات مهم در این مثال:

از پارامتر random_state در ورودی‌های تابع train_test_split، به عنوان یک seed برای انجام درهم‌سازی داده‌ها پیش از تقسیم استفاده می‌شود. مقدار none برای این پارامتر، به معنای درنظر گرفتن مقدار رندم برای seed و انجام رندم درهم‌سازی در هر فراخوانی است.

خروجی تابع train_test_split چهار مولفه دارد. مولفه اول و سوم به ترتیب برابر مجموعه داده‌ی آموزش و برچسب‌های آن رکوردها هستند. مولفه دوم و چهارم نیز به ترتیب نظیر مجموعه داده‌ی تست و برچسب‌های رکوردهای واقع‌شده در این زیرمجموعه هستند.

```
model.fit(X_train, y_train, epochs=100, batch_size=10)
```

با تقسیم داده، رکوردهای مورداستفاده در فراخوانی تابع fit زیرمجموعه‌ی مربوط به train خواهد بود. یعنی مشابه مثال بالا دیتاست ورودی به این تابع X_train خواهد بود.

```
model.evaluate(X_train, y_train, batch_size=None, verbose=0)
model.evaluate(X_test, y_test, batch_size=None, verbose=0)
```

برای مشاهده‌ی میزان loss در هر یک از گام‌های آموزش و تست می‌توان تابع evaluate را با ورودی‌های نظیر دیتاست‌های آموزش و تست فراخوانی کرد.

```
model.fit(X, y, epochs=100, batch_size=10, validation_split=0.3)
```

همان‌طور که پیش از این نیز بررسی شده است، راه دیگر تقسیم داده‌ها استفاده از پارامتر validation_split است. ورودی این پارامتر مشخص‌کننده‌ی این است که چند درصد از رکوردهای آخر دیتاست برای تست انتخاب شوند. برای نمونه در این مثال ۷۰ درصد از رکوردهای اول دیتاست به عنوان داده‌ی آموزش و ۳۰ درصد آخر به عنوان داده‌ی تست مورد استفاده قرار خواهند گرفت.

Bias و Variance در مدل‌ها

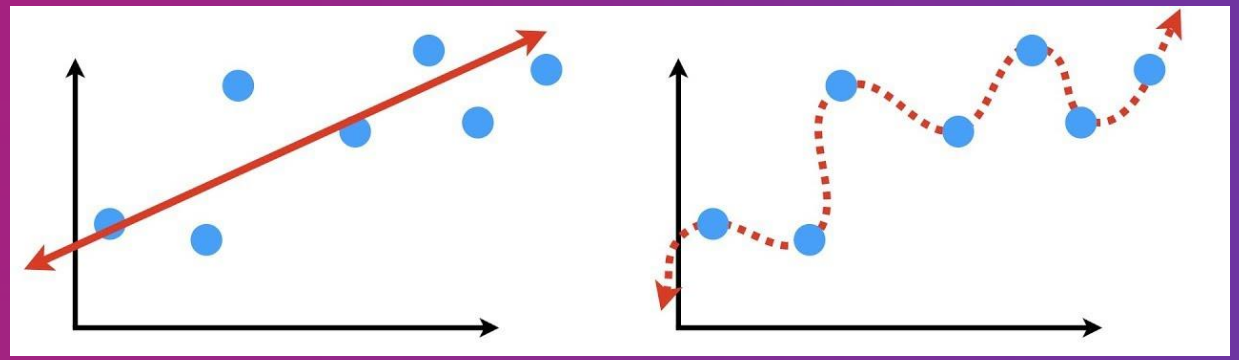
Bias: اختلاف بین میانگین پیش‌بینی‌های مدل و مقادیر واقعی را bias می‌نامند.

مدل با bias بالا: مدلی که به اندازه‌ی کافی انعطاف‌پذیر نبوده و روابط بین داده‌ها را به خوبی نیاموزد، دارای خطای زیادی در گام آموزش خواهد بود. در این مدل‌ها مقدار bias زیاد خواهد بود. در حقیقت این مدل توجه کافی را به داده‌های آموزش نکرده است. در این مدل‌ها اختلاف bias و variance کم خواهد بود.

Variance: تنوع و اختلاف نتایج مدل در مقادیر خطا را variance می‌نامند.

مدل با variance بالا: مدلی که نسبت به یک دیتاست خاص، بیش از حد انعطاف‌پذیری داشته باشد و علاوه بر روابط بین داده‌ها، نویزها را هم مورد آموزش قرار دهد دارای خطای بالایی در گام تست خواهد بود. در این مدل اختلاف خطای آموزش و تست زیاد و اصطلاحاً مدل دارای variance زیادی است.

نکته: مدل ایده‌آل دارای حداقل bias و variance است.

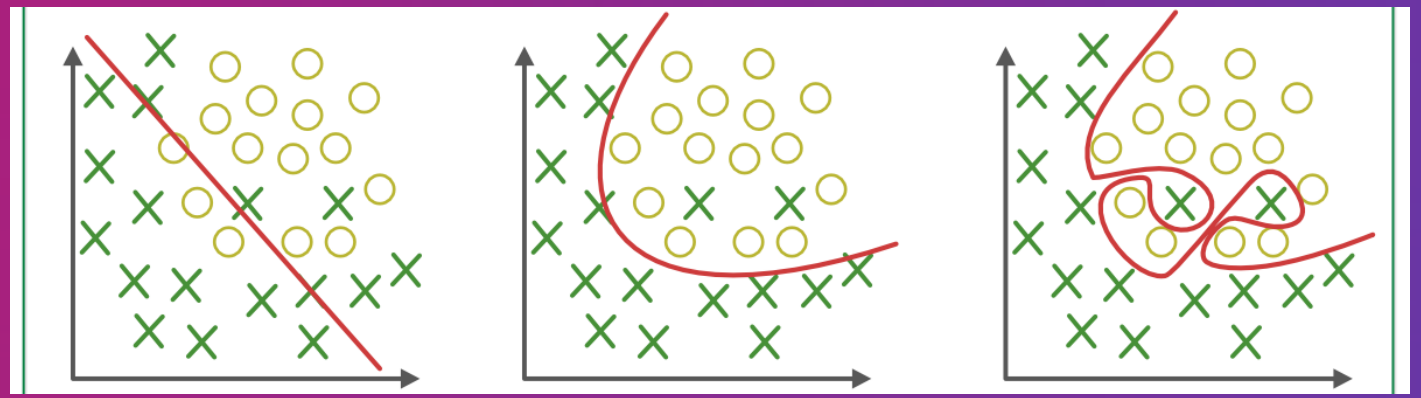


Underfitting و Overfitting

Underfitting: یک مدل بیش از حد ساده، قادر به پیش‌بینی دیتاست‌های پیچیده نخواهد بود. چنین مدلی دچار مشکل underfiiting خواهد شد. یعنی مدل تقریب‌هایی ساده‌تر از واقعیت خواهد زد. مدل با bias بالا دچار underfitting است.

Overfitting: مدل‌هایی که انعطاف‌پذیری و پیچیدگی بالاتری از توزیع واقعی داده‌ها داشته باشند، دچار مشکل overfitting خواهند شد. افزایش پیچیدگی با افزایش بیش از حد لایه‌ها و یا گره‌ها رخ خواهد داد. در این حالت، آموزش بهتر از حالت underfitting انجام می‌شود، اما خطای تست بسیار بالا است. زیرا مدل بیش از حد بر داده‌های دیده‌شده تمرکز کرده است. مدل‌های دارای variance بالا دچار overfitting خواهند شد.

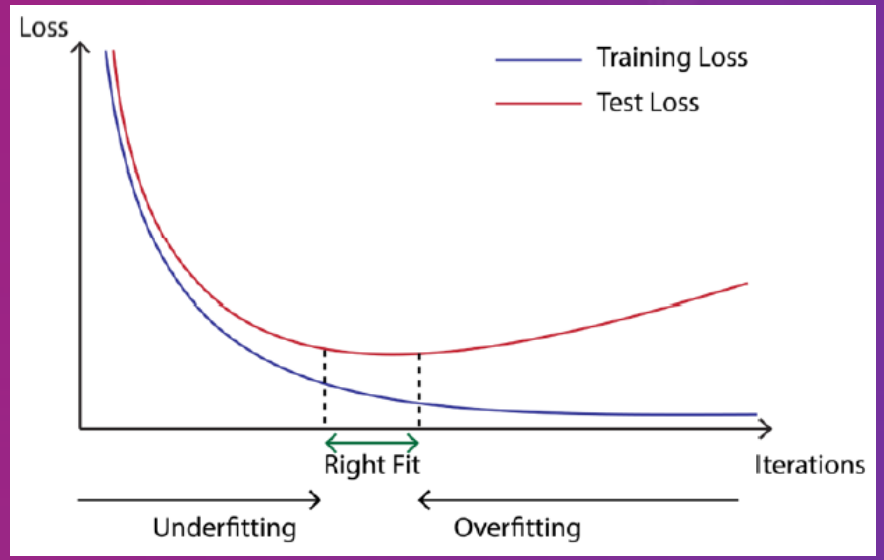
برای انتخاب بهترین مدل، باید حالت‌های رخداد overfit و underfit را مقایسه و مناسب‌ترین سطح پیچیدگی را برای مدل انتخاب کرد.



Early Stopping

یکی دیگر از دلایل overfitting تکرار بیش از حد کم (زیاد) iteration ها است.

Early Stopping: راه حل مقابله با overfit در اثر تکرار زیاد الگوریتم، روشی به نام early stopping است. در این روش با بررسی خطای آموزش و خطای تست در هر تکرار، محل شروع overfit شناسایی می‌شود. در نتیجه اگر تکرار الگوریتم را در لحظه‌ی شروع overfitting خاتمه دهیم، مدل دچار overfit نخواهد شد.



Early Stopping

چندین برنامه برای بررسی خطای test و train:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/Training_and_Test_Loss.ipynb

در این مثال نحوه‌ی دریافت و ذخیره‌ی خطای validation در حین هر epoch را یاد خواهیم گرفت. از خطاهای ذخیره‌شده برای تشخیص epoch منجر به overfit استفاده می‌شود.

دستورات مهم در این مثال:

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, batch_size=10)
```

همان‌طور که گفته شد یک روش برای تقسیم دیتاست، استفاده از تابع train_test_split است. در این حالت می‌توان با پارامتر validation_data در تابع fit دیتاست‌های مربوط به validation را مشخص کرد. خروجی دستور فوق خطای آموزش و validation در هر epoch خواهد بود.

```
model.fit(X, y, validation_split=0.3, epochs=100, batch_size=10)
```

با استفاده از پارامتر validation_split نیز می‌توان خطا را پس از پایان هر epoch دریافت کرد.

فعالیت 3.02: تشخیص بیماری Fibrosis با شبکه عصبی

نمونه برنامه‌ای برای تمرین پیاده‌سازی یک شبکه عصبی و بررسی خطاهای مدل:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%203/Activity3_02.ipynb

در این فعالیت پیش بینی خواهیم کرد که یک فرد براساس مشخصاتی چون سن، جنسیت و شاخص BMI دارای fibrosis است یا خیر. بنابراین مسئله دو کلاسه خواهد بود، به طوری که فرد سالم دارای برچسب صفر و فرد مریض دارای برچسب یک است. دیتاست شامل ۱۳۸۵ بیمار و ۲۸ تا ویژگی برای هر فرد است. فایل "HCV_feats.csv" شامل دیتاست و فایل "HCV_target.csv" حاوی برچسب نظیر رکوردهایمان می‌باشد.

در این فعالیت با رسم نمودارهایی مربوط به خطای train و test، محل رخداد overfitting را تشخیص داده و موثرترین تعداد epoch را پیدا خواهیم کرد. طبق موارد خواسته‌شده فعالیت را کامل کنید.

دستورات مهم در این فعالیت:

```
from sklearn.preprocessing import StandardScaler
```

از تابع StandardScaler برای استانداردکردن ویژگی‌ها استفاده می‌شود. در خروجی این تابع ویژگی‌ها دارای میانگین صفر و واریانس یکسان برای هر رکورد خواهند بود.

فعالیت 3.02: تشخیص بیماری Fibrosis با شبکه عصبی

دستورات مهم در این فعالیت:

```
X = pd.DataFrame(sc.fit_transform(X), columns=X.columns)
```

با تابع DataFrame داده به صورت label دار (در سطر و ستون) در pandas ساخته خواهد شد. با فراخوانی fit_transform ابتدا با تابع fit از pandas واریانس یکسان محاسبه خواهد شد. سپس با transform داده‌ها به نحوی تغییر خواهند کرد که در fit (واریانس یکسان) صدق کنند. با پارامتر columns=X.columns نیز مشخص کردیم که برچسب‌های ستون‌های داده‌ی استانداردشده، همان برچسب‌های ستون‌های X باشند.

```
print(f"Best Accuracy on training set = {max(history.history['accuracy'])*100:.3f}%")
```

در پایتون نسخه‌ی سه به جای چاپ به روش % formatting که پیش از این داشتیم، از روش f-string formatting استفاده می‌شود. در این روش رشته با حرف f شروع شده و متغیرها داخل {} قرار می‌گیرند. با مقدار 3f. نیز تعداد سه رقم اعشار مشخص می‌گردد. پس در این جا حداکثر مقدار میان دقت تمام epoch ها به درصد (تا سه رقم اعشار) به عنوان متغیر در رشته چاپ خواهد شد.

خلاصه

صفحه ۱۱۸

دوره‌ی آموزشی یادگیری عمیق با Keras

خلاصه

در این فصل با مفاهیم زیر آشنا شدیم:

- روش forward propagation
- Loss function به عنوان معیاری برای سنجش کارایی
- الگوریتم gradient descent
- Backpropagation و محاسبه‌ی Loss نسبت به پارامترهای مدل
- تقسیم دیتاست و ساخت مجموعه‌های validation و test
- تحلیل خطاهای مدل و تشخیص overfitting و underfitting
- نحوه‌ی پیاده‌سازی شبکه‌های عصبی تک لایه و چند لایه

پاییز ۱۳۹۹

ارزیابی مدل با cross-validation به وسیله‌ی wrappers

فصل چهارم

دوره‌ی آموزشی یادگیری عمیق با Keras

مقدمه

صفحه ۱۲۲

مقدمه

در این فصل با کتابخانه‌ی scikit-learn آشنا شده و به کمک آن یک wrapper در keras خواهید ساخت. همچنین برای ارزیابی مدل‌ها از cross-validation استفاده خواهید کرد.

در فصل قبل میزان کارایی مدل‌های مختلف را براساس loss مورد بررسی قرار دادیم. و براساس همین پارامتر، از رخداد overfitting جلوگیری کردیم. در این فصل با یکی از روش‌های نمونه‌برداری مجدد (resampling) به نام cross-validation بهترین پارامترها را پیدا خواهیم کرد. استفاده از cross-validation باعث افزایش دقت و رسیدن به مدلی قوی و پایدار (robust) خواهد شد.

در بخش دیگری از فصل چهارم، در مورد علت استفاده از CV، انواع آن و تفاوت‌های آن‌ها با یکدیگر صحبت خواهیم کرد.

فایل‌های مورد نیاز در فصل چهارم

لینک دانلود کد تمامی تمرین‌ها و فعالیت‌های این فصل:

https://github.com/ymgh96/Keras-Programming-Course/tree/main/Chapter%204

لینک دانلود دیتاست‌های مورد استفاده در این فصل:

https://github.com/ymgh96/Keras-Programming-Course/tree/main/Chapter%204/data

Cross Validation

صفحه ۱۲۲

Cross-Validation

Resampling: در اکثر روش‌های resampling رکوردها به صورت تکرارشونده از دیتاست انتخاب می‌شوند، تا چندین حالت برای زیرمجموعه‌های آموزش و تست ساخته شوند. سپس در هر تکرار، بر یک حالت از زیرمجموعه‌ها یادگیری و ارزیابی انجام خواهد شد. نتایج روش‌های resampling کاراتر از یک بار تکرار آموزش و ارزیابی خواهد بود.

یکی از مهم‌ترین و پرکاربردترین روش‌های resampling، cross-validation است. CV یکی از پراستفاده‌ترین روش‌ها برای دیتاست‌های کوچک و حاوی تعداد کم داده است.

انواع CV:

- K-fold cross-validation
- Leave-one-out cross-validation

یک‌بار تقسیم دیتاست

در روشی که در فصل قبل آموختیم، بخشی از دیتاست به صورت تصادفی به عنوان زیرمجموعه‌ی تست و باقی رکوردها به عنوان زیرمجموعه‌ی آموزش درنظر گرفته می‌شدند.

مزایای این روش:

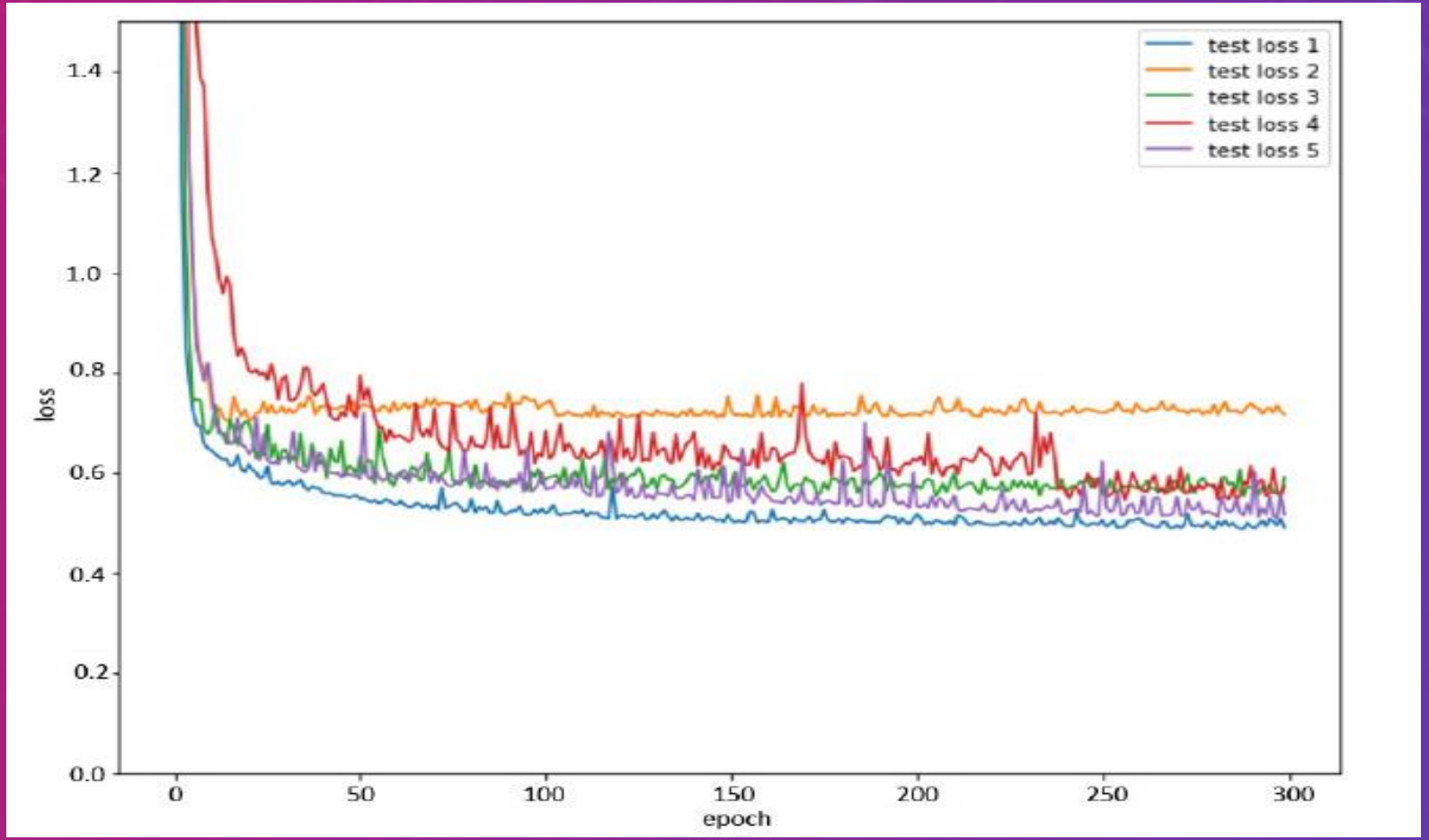
- سادگی
- پیاده‌سازی ساده
- محاسبات کم‌هزینه

معایب این روش:

- وابستگی بالای خطای تست به داده‌های انتخاب‌شده به عنوان زیرمجموعه تست
- عدم شرکت گروهی خاص از داده‌ها (زیرمجموعه‌ی تست) در فرآیند یادگیری

در مقابل این روش، در روش cross-validation تمامی داده‌ها در آموزش شرکت خواهند کرد.

یک‌بار تقسیم دیتاست



K-Fold Cross-Validation

K-fold CV: در این روش، دیتاست به k تا زیرمجموعه (fold) تقریباً هم اندازه تقسیم می‌شود.
در هر تکرار از الگوریتم یک fold به عنوان مجموعه‌ی تست و k-1 بخش دیگر برای آموزش استفاده می‌شوند.
بنابراین الگوریتم k بار تکرار و خطای مدل، میانگین خطای هر یک از این k مرتبه خواهد بود.
مقادیر مرسوم برای k، ۵ و ۱۰ است.

Leave-one-out Cross-Validation

Leave-one-out (LOO) CV: در این روش در هربار تکرار تنها یکی از داده‌ها برای تست انتخاب خواهد شد و مابقی داده‌ها برای آموزش استفاده خواهند شد. بنابراین تعداد تکرارهای این الگوریتم به تعداد داده‌های دیتاست است و خطای مدل، میانگین خطای تمام تکرارها خواهد بود.

معایب:

- خطای هر تکرار وابسته به یک داده (زیرمجموعه‌ی تست) است و variance تکرارها بسیار زیاد خواهد بود.
- تعداد تکرارها بسیار زیاد خواهد بود.

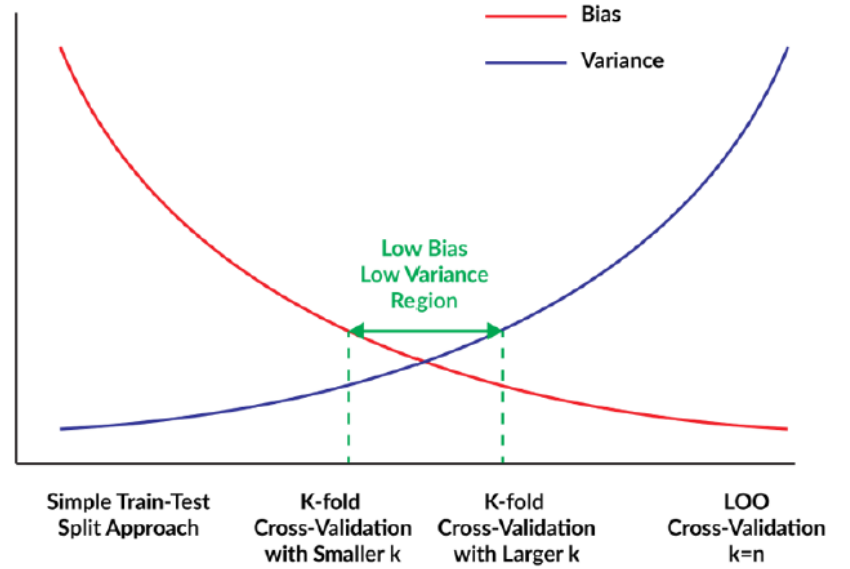
مزایا:

- برخلاف تقسیم یکباره‌ی دیتاست، در هر تکرار اکثر رکوردهای دیتاست در یادگیری شرکت خواهند کرد.
- انتخاب تصادفی در قرارگیری داده‌ها در هر بخش، برای هر تکرار وجود نداشته و هر داده به یک حالت در تست شرکت داده می‌شود.

مقایسه‌ی k-fold و LOO

الگوریتم LOO حالتی خاص از k-fold است. به بیان دیگر اگر در k-fold مقدار k را برابر سایز دیتاست درنظر بگیریم، روند الگوریتم k-fold معادل LOO خواهد بود.

- هزینه‌ی اجرای k-fold با k=5, 10 بسیار کمتر از حالت k=n است.
- روش k-fold نسبت به LOO دارای variance کمتری است.
- روش LOO نسبت به k-fold دارای bias کمتری است.



Cross Validation برای مدل‌های یادگیری عمیق

صفحه ۱۲۹

ساخت wrapper در keras با scikit-learn

Wrapper: wrapper ها توابعی هستند که امکان فراخوانی توابع و کتابخانه‌های دیگر را به صورت encapsulate شده در کد برنامه‌نویس فراهم می‌کنند. در حقیقت wrapper یک interface برای استفاده از API های دیگر در کد برنامه‌نویس است.

تابع cross-validation در کتابخانه‌ی scikit-learn قرار دارد، پس برای فراخوانی آن در کد مبتنی بر keras لازم است تا یک واسط (wrapper) در keras تعریف کنیم تا بتوانیم با این کتابخانه کار کنیم.

keras.wrappers.scikit_learn.KerasClassifier(build_fn=None, **sk_params)

با کد خط بالا یک interface از scikit-learn در keras برای classification (به طور مشابه برای regression) ساخته می‌شود. ورودی این wrapper یک تابع حاوی بدنه‌ی مدل و سایر پارامترهای یادگیری (batch_size، تعداد epoch ها و...) خواهد بود.

تمرین 4.01: ساخت wrapper با scikit-learn برای مسال رگرسیون

نمونه برنامه‌ای برای تمرین کار با wrapper و cross-validation:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%204/Exercise4_01.ipynb

هدف مسئله پیش‌بینی میزان سمی‌بودن (متغیری پیوسته) مواد شیمی بوده و بنابراین مسئله رگرسیون است. دیتاست ما شامل ۹۰۸ رکورد و ۶ تا مشخصه از مواد شیمیایی است. مقدار برچسب نیز ستونی با نام LC50 است.

دستورات مهم در این تمرین:

```
data = pd.read_csv('../data/qsar_fish_toxicity.csv', sep=';', names=colnames)
```

با تابع read_csv در panda می‌توان یک فایل اکسل را خوانده و محتوای آن را در یک ماتریس ریخت. خروجی این دستور یک داده از نوع dataframe است، یعنی می‌توان برای ستون‌های آن label تعریف کرد. اگر ستون‌های فایل خوانده‌شده دارای برچسب باشند، همان برچسب‌ها برای ماتریس خروجی نیز درنظر گرفته می‌شوند و در غیر این‌صورت، مشابه مثال بالا با پارامتر names می‌توان مقادیر برچسب‌ها را خودمان مشخص کنیم. در این مثال، مقادیر یک آرایه (colnames) به عنوان برچسب ستون‌ها درنظر گرفته شده است. از پارامتر sep به عنوان deliamtor در تعیین معیار جداکننده‌ی داده‌ها است.

```
X = data.drop('LC50', axis=1)
```

به کمک تابع drop از panda می‌توان یک سطر و یا ستون را براساس label و یا index از یک ماتریس حذف کرد. در این مثال ستونی با نام (برچسب) LC50 از ماتریس data حذف خواهد شد. برای مشخص‌کردن اینکه حذف از سطرها انجام گیرد یا از ستون‌ها، مقدار پارامتر axis را به ترتیب باید صفر و یا یک مقداردهی کرد.

تمرین 4.01: ساخت wrapper با scikit-learn برای مسال رگرسیون

دستورات مهم در این تمرین:

```
def build_model():
    …
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

برای استفاده از wrapper مربوط به scikit-learn، لازم است تا بدنه‌ی مدل در قالب یک تابع تعریف و ساخته شود. به همین منظور در این مثال یک تابع به نام build_model تعریف کرده و لایه‌های مدل را مشخص، مدل را کامپایل و درنهایت مدل ساخته‌شده را به عنوان خروجی این تابع برمی‌گردانیم.

```
from keras.wrappers.scikit_learn import KerasRegressor
YourModel = KerasRegressor(build_fn= build_model, epochs=100, batch_size=20, verbose=1)
```

برای تعریف wrapper از کتابخانه‌ی keras.wrappers.scikit_learn استفاده می‌شود. برای مسائل دسته‌بندی از تابع KerasClassifier از این کتابخانه، و برای مسائل رگرسیون از تابع KerasRegressor استفاده می‌شود. در فراخوانی wrapper ها، در پارامتری به نام build_fn نام تابعی که بدنه‌ی مدل در آن ساخته شده است را مشخص می‌کنیم. همچنین سایر پارامترهای یادگیری مانند تعداد epoch ها و اندازه‌ی batch ها را نیز به عنوان ورودی سازنده‌ی wrapper وارد می‌کنیم.

تمرین 4.01: ساخت wrapper با scikit-learn برای مسال رگرسیون

دستورات مهم در این تمرین:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(YourModel, X, y, cv=5)
```

پس از ساخت wrapper از scikit learn می‌توان از توابع این کتابخانه در keras استفاده کرد. برای استفاده از تابع cross validation از این کتابخانه، از زیرمجموعه‌ی model_selection تابع cross_val_score را import می‌کنیم. ورودی این تابع wrapper ساخته‌شده، دیتاست، برچسب‌های نظیر رکوردهای دیتاست و نوع cross validation موردنظر به عنوان مقدار پارامتر cv می‌باشد. اگر ورودی پارامتر cv را عدد قرار دهیم، الگوریتم مورداستفاده به صورت پیش‌فرض stratified K-fold CV بوده و مقدار k برابر عدد واردشده خواهد بود. خروجی تابع cross_val_score مقدار پارامترهای مشخص‌شده (به صورت پیش‌فرض loss) برای هر یک از تکرارها (fold ها) می‌باشد.

```
scores = cross_val_score(YourModel, X, y, cv=5, scoring='neg_mean_absolute_error')
```

خروجی تابع cross_val_score براساس metric های تعریف‌شده در مدل محاسبه می‌شود. اما برای تغییر آن می‌توان از پارامتر scoring استفاده کرده و آن را بسته به مسئله مقداردهی کرد.

تمرین 4.01: ساخت wrapper با scikit-learn برای مسال رگرسیون

دستورات مهم در این تمرین:

```
scores = cross_val_score(YourModel, X, y, cv=LeaveOneOut())
```

برای استفاده از الگوریتم LeaveOneOut در فراخوانی تابع cross_val_score، مقدار پارامتر cv را برابر LeaveOneOut() قرار می‌دهیم.

```
print(scores.mean())
```

از آن‌جا که خطای روش‌های CV برابر میانگین خطاهای بدست‌آمده در تمامی تکرارهای این الگوریتم است، در نهایت نیز به کمک تابع mean بر تمامی خطاهای بدست‌آمده از اجرا میانگین می‌گیریم.

پیاده‌سازی‌های CV در scikit learn

- KFold(n_splits=?): این روش در حقیقت پیاده‌سازی حالت سنتی k-fold CV است. در این حالت تعداد fold ها برابر مقدار n_splits درنظر گرفته می‌شود.
- RepeatedKFold(n_splits=?, n_repeats=?): در این روش الگوریتم k-fold (k=n_splits)، n_repeats مرتبه تکرار خواهد شد.
- LeaveOneOut(): این تابع معادل الگوریتم سنتی LOO می‌باشد.
- ShuffleSplit(n_splits=?): در این تابع، الگوریتم k-fold به همراه درهم‌ریختگی (shuffle) اجرا می‌شود. در این روش قبل از هر تکرار، ابتدا داده‌ها درهم ریخته و سپس $\frac{1}{k}$ از آن‌ها برای تست و مابقی داده‌ها برای آموزش استفاده خواهند شد. بنابراین در این روش برخلاف الگوریتم k-fold سنتی، ممکن است یک داده چندین بار در تست شرکت کند.
- حالت‌های طبقه‌بندی شده (stratified): تمامی موارد گفته‌شده در بالا دارای یک نسخه پیاده‌سازی به حالت stratified نیز هستند. این حالت مناسب زمانی است که داده‌های کلاس‌های مختلف دارای توزیع متوازنی نباشند. برای مثال در قطعه کد زیر روش k-fold (n_splits) در حالت stratified مورد استفاده قرار گرفته است.

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5)
scores = cross_val_score(YourModel, X, y, cv=skf)
```

تمرین 4.02: ارزیابی شبکه‌های عصبی عمیق با CV

نمونه برنامه‌ای برای مرور استفاده از cross-validation در ارزیابی:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%204/Exercise4_02.ipynb

در این تمرین قصد داریم تا در مسئله‌ی پیش‌بینی میزان سمی‌بودن مواد شیمی از تابع Kfold از کتابخانه‌ی model_selection استفاده کرده و به این‌صورت مقدار پارامتر cv را در فراخوانی cross_val_score تعیین کنیم.

دستورات مهم در این تمرین:

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5)
results = cross_val_score(YourModel, X, y, cv=kf)
```

همان‌طور که پیش از این نیز گفته شد، الگوریتم‌های تعریف‌شده در scikit learn باید به عنوان ورودی پارامتر cv در فراخوانی cross_val_score مشخص شوند. برای مثال در کد بالا یک نمونه از Kfold با k=5 تعریف و در شی kf ریخته شده است. سپس kf به عنوان مقدار پارامتر cv تعیین شده است.

تمرین 4.02: ارزیابی شبکه‌های عصبی عمیق با CV

دستورات مهم در این تمرین:

```
print(f"Final Cross Validation Loss = {abs(results.mean()):.4f}")
```

در keras خروجی خطا (results) به صورت صعودی اعلام می‌شود. یعنی هر چقدر مقدار بزرگ‌تر باشد، نتیجه بهتر است. بنابراین خطای mean_squared_error به صورت مقادیر منفی بوده و ما برای چاپ از تابع قدر مطلق استفاده می‌کنیم.

در این جا چاپ رشته به روش f-string formatting انجام شده است. یعنی بر میانگین خطای k دور اجرای cv، قدرمطلق اعمال‌شده و سپس در انتهای رشته تا چهار رقم اعشار نمایش داده می‌شود.

فعالیت 4.01: ارزیابی مدل با CV برای مسئله‌ی تشخیص بیماری Fibrosis

فعالیتی برای تعریف wrapper در keras و استفاده از cross validation:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%204/Activity4_01.ipynb

در این فعالیت قصد داریم تا روش cross validation را بر مسئله‌ی تشخیص بیماری Fibrosis پیاده کنیم. این دیتاست شامل ۱۳۸۵ رکورد و ۲۸ تا ویژگی بود. مسئله دو کلاسه و برچسب‌ها دارای دو مقدار صفر و یک هستند. مقدار صفر برای برچسب به معنای سالم‌بودن و مقدار یک به معنای مبتلا بودن فرد است.

در فصل سه ارزیابی بر این دیتاست را با روش تقسیم یکباره‌ی داده‌ها برای آموزش و تست، انجام دادیم. در این فصل قصد داریم تا ارزیابی را با روش CV انجام دهیم.

با توجه به گام‌های تعریف‌شده در کد این فعالیت، آن را تکمیل کنید.

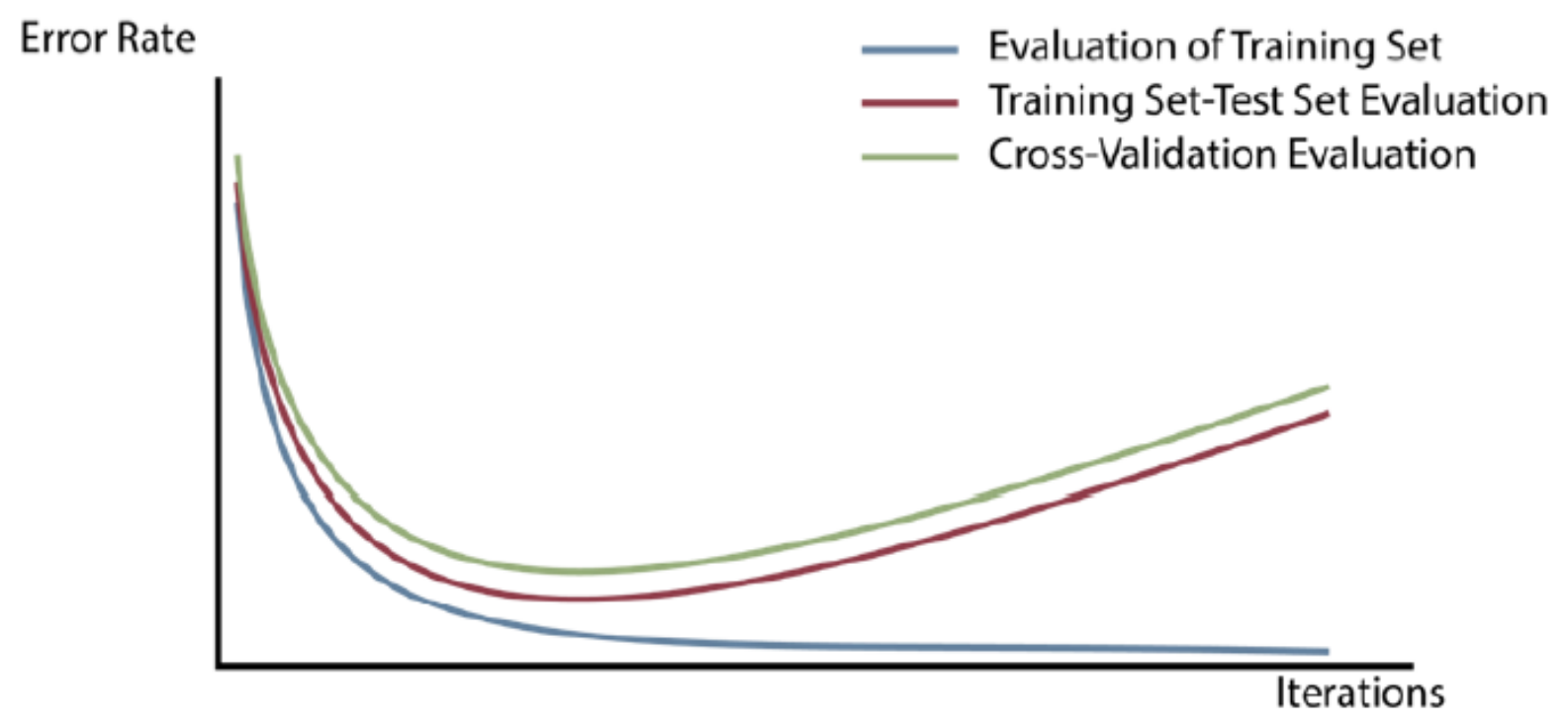
انتخاب مدل با Cross-Validation

صفحه ۱۴۰

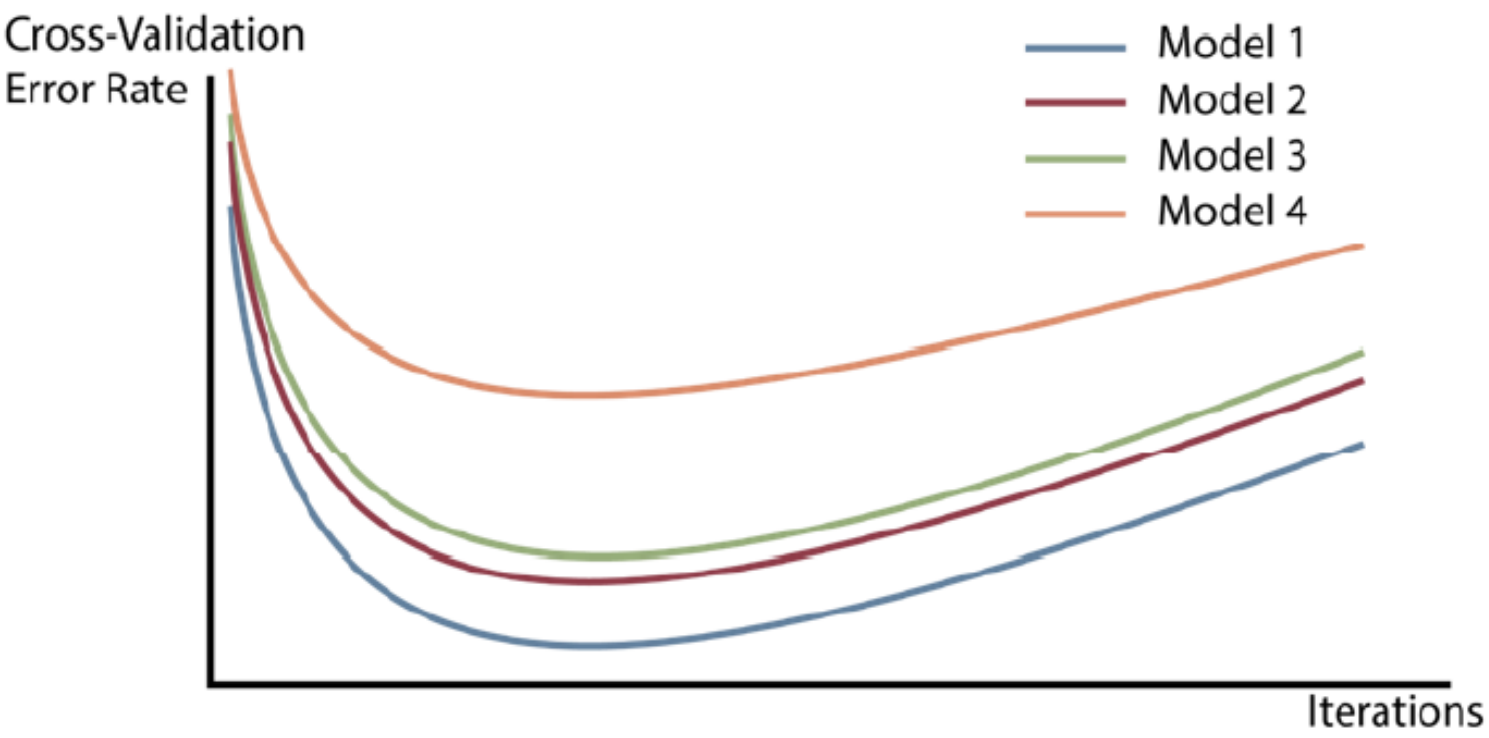
انتخاب مدل با Cross-Validation

همان‌طور که تا این‌جا مشخص شد، به کمک روش CV می‌توان مدل‌ها را مورد ارزیابی قرار داده و خطای هر یک را بدست آورد. حال اگر مدل‌های مختلفی را با پارامترهای مختلف توسط روش CV مورد ارزیابی قرار دهیم، با مقایسه‌ی خطاهای بدست آمده می‌توان مدل با کمترین خطا را انتخاب کرد. بنابراین برای یک مسئله می‌توان با بررسی خطا بر حالات مختلف مدل‌ها، بهترین مدل و بهترین hyperparameter ها را انتخاب کرد.

CV برای ارزیابی و انتخاب مدل



CV برای ارزیابی و انتخاب مدل



تمرین 4.03: نوشتن توابع برای پیاده‌سازی مدل‌های یادگیری عمیق با cross-validation

نمونه برنامه‌ای برای انتخاب بهترین مدل و hyperparameter ها با استفاده از cross-validation:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%204/Exercise4_03.ipynb

در این تمرین به کمک ارزیابی با cross-validation، بین سه مدل با معماری‌های مختلف، حالت‌های متفاوت برای hypertparameter های activation function، تعداد گره‌های لایه‌های مدل، تعداد epoch، اندازه‌ی batch و الگوریتم optimizer انتخاب خواهیم کرد.

دیتاست در این سوال مربوط به مواد شیمیایی است. همان‌طور که در مثال‌های قبل نیز داشتیم، این دیتاست دارای ۹۰۸ رکورد و ۶ ویژگی بود.

دستورات مهم در این تمرین:

```
models = [build_model_1, build_model_2, build_model_3]

for m in range(len(models)):

    model = KerasRegressor(build_fn=models[m], epochs=100, batch_size=20, verbose=0, shuffle=False)
```

برای مقایسه‌ی مدل‌های مختلف با هم، بدنه‌ی هر مدل را در تابعی مجزا تعریف می‌کنیم. سپس در یک حلقه هر کدام از مدل‌ها (models[m]) به عنوان مقدار نظیر build_fn مشخص می‌کنیم.

تمرین 4.03: نوشتن توابع برای پیاده‌سازی مدل‌های یادگیری عمیق با cross-validation

دستورات مهم در این تمرین:

```
epochs = [100, 150]
batches = [20, 15]
```

برای بررسی خطا بر پارامترهای مختلف، مقادیر موردنظر برای پارامترها را در آرایه قرار می‌دهیم. به این ترتیب با اجرای حلقه بر هر یک، می‌توان یادگیری را بر اساس هر کدام انجام داد.

```
for e in range(len(epochs)):
    for b in range(len(batches)):
        model = KerasRegressor(build_fn= build_model_2, epochs= epochs[e], batch_size= batches[b], verbose=0, shuffle=False)
```

در مثال بالا قصد داریم تا بهترین مقادیر را برای دو پارامتر تعداد epoch ها و اندازه‌ی batch بدست آوریم. پارامترها برای بهترین مدل (build_model_2) محاسبه خواهند شد. بنابراین درون دو حلقه تمامی حالت‌های مدنظر برای این دو پارامتر را مورد یادگیری قرار داده و خطای هر حالت را برای مقایسه نگه می‌داریم.

فعالیت 4.02: انتخاب مدل با Cross-Validation برای تشخیص بیماری Fibrosis

فعالیتی برای انتخاب بهترین مدل و hyperparameter ها با استفاده از cross-validation:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%204/Activity4_02.ipynb

در این فعالیت به کمک ارزیابی با cross-validation، بین سه مدل با معماری‌های مختلف و حالت‌های متفاوت برای hypertparameter های activation function، تعداد epoch، اندازه‌ی batch و الگوریتم optimizer انتخاب خواهیم کرد.

دیتاست مورد استفاده در این فعالیت مربوط به تشخیص بیماری Fibrosis است که پیش از این نیز بررسی کردیم.

با توجه به گام‌های تعریف‌شده در کد مربوط به فعالیت، آن را کامل کنید.

فعالیت 4.03: انتخاب مدل با Cross-Validation برای دیتاست میزان ترافیک

فعالیتی برای انتخاب مدل و hyperparameter ها با استفاده از cross-validation:

https://github.com/ymgh96/Keras-Programming-Course/blob/main/Chapter%204/Activity4_03.ipynb

در این فعالیت به کمک ارزیابی با cross-validation، بین سه مدل با معماری‌های مختلف، حالت‌های متفاوت برای hypertparameter های activation function، تعداد epoch، اندازه‌ی batch و الگوریتم optimizer انتخاب خواهیم کرد.

دیتاست در این فعالیت شامل اطلاعات ترافیکی خودروها به صورت نرمال شده است.

هدف در این فعالیت پیش‌بینی حجم ترافیک خودروها در ساعت است. بنابراین مسئله رگرسیون بوده و قصد داریم تا بهترین مدل را در مسائل رگرسیون نیز انتخاب کنیم.

دستورات مهم در این فعالیت:

```
from keras.wrappers.scikit_learn import KerasRegressor

regressor = KerasRegressor(build_fn=build_model_2, epochs=epochs[i], batch_size=batches[j], verbose=0, shuffle=False)
```

در مسائل رگرسیون، برای تعریف wrapper از scikit learn باید از تابع KerasRegressor استفاده کرد.

خلاصه

صفحه ۱۵۲

دوره‌ی آموزشی یادگیری عمیق با Keras

خلاصه

در این فصل با مفاهیم زیر آشنا شدیم:

- مفهوم resampling
- روش cross-validation برای ارزیابی
- انواع cross-validation
- Wrapper ها در keras و استفاده از کتابخانه‌ی scikit learn در keras
- نحوه‌ی انتخاب بهترین مدل و پارامترها به کمک cross validation