پروژه اول شبکه عصبی

عنوان پروژه: پیش‌بینی دیابت روی دیتاست Pima Indians به کمک شبکه MLP.

هدف اصلی پروژه: آشنایی عملی با پیاده‌سازی‌های شبکه پرسپترون چندلایه (MLP).

سایر اهداف پروژه: آشنایی عملی با پیش پردازش داده‌های پزشکی، مقایسه توابع فعالسازی، آشنایی مفهومی و عملی با شبکه KAN و مقایسه آن با شبکه MLP.

مجموعه دادگان: دیتاست Pima Indians Diabetes (۸ ویژگی عددی + برچسب دودویی دیابت) با اندازه نمونه ۷۶۸.

پیش‌نیازها:

- Python +3.9
- کتابخانه‌های torch، matplotlib (پلات کردن داده‌ها)، scikit-learn (خواندن داده‌ها)، numpy و pandas
- کتابخانه pykan (بخش امتیازی)

پروتکل ارزیابی:

- تقسیم‌بندی داده ها به صورت train/val/test = 60/20/20.
- پیش پردازش
  - جایگزینی مقدار نامعتبر/صفرهای غیرواقعی احتمالی در برخی ویژگی‌ها و سپس میانگین‌گذاری و یا میانه‌گذاری بر حسب کلاس (یا SimpleImputer).
  - StandardScaler روی ویژگی‌ها
- معیارهای ارزیابی عبارت‌اند از Accuracy, F1-Score, ROC-AUC. در صورت عدم توازن کلاس می‌توان از WeightedSampler و یا class_weight برای وزن‌دهی کلاس‌ها استفاده نمود.
- برای کنترل تکرارپذیری مقدار seed برای numpy و torch تنظیم و ثابت شود (مثلا ۴۲).
- در صورت مشاهده overfit از تکنیک Early Stop با Patience مناسب استفاده شود.
- گزارش هر فاز به صورت یک جدول (شامل مقادیر برای متریک‌های ذکر شده) و یک نمودار شامل دو منحنی loss برای train و test بر حسب epoch. استفاده از Confusion Matrix در گزارش نیز مجاز می‌باشد اما الزامی نیست.

فاز اول (پیاده‌سازی MLP با Pytorch)

در این فاز باید یک شبکه MLP با استفاده از کتابخانه Pytorch به عنوان Baseline استفاده شود. برای توابع فعال‌سازی از Relu و تابع خروجی از Sigmoid استفاده شود.

نکات تکمیلی:

- مقدار Batch size و تعداد epoch باید به‌گونه‌ای تنظیم شوند که بهبود نتیجه train و تست نهایی کمک کنند. در این قسمت حداقل 90 درصد دقت train و 80 درصد دقت test انتظار می‌رود.
- استفاده از بهینه‌ساز Adam (مثلا با نرخ یادگیری 0.001) مجاز می‌باشد.
- در این فاز باید نوت بوک Phase1.ipynb که در اختیار قرار گرفته تکمیل شود.

فاز دوم (پیاده‌سازی MLP با MyTorch)

در فاز اول با پیاده‌سازی شبکه‌های عصبی به کمک کتابخانه PyTorch آشنا شدیم. حال در فاز دوم هدف این است که عمیق‌تر وارد این حوزه شویم و خودمان یک کتابخانه برای انجام این کار پیاده‌سازی کنیم که MyTorch نام دارد. در این فاز باید اجزای مختلف یک شبکه عصبی از پایه طراحی شوند تا نحوه عملکرد درونی آن‌ها به‌خوبی درک شود. سپس شبکه MLP به کمک این کتابخانه (و بدون استفاده از کتابخانه‌های دیگر نظیر Pytorch و Tensorflow) آموزش داده شده و تست شود. در ادامه جزئیات مربوط به پیاده سازی کتابخانه MyTorch آمده است.

۱. کلاس Tensor

این کلاس اساس کار ما است و تمامی محاسبات پایه در شبکه عصبی باید بر اساس آن انجام گیرد.
یکی از فیلدهای مهم این کلاس باید از نوع numpy.ndarray باشد. در واقع تمام عملیات ماتریسی باید با این نوع داده انجام شود.
پیشنهاد می‌شود پیش از شروع پروژه فایل tensor.py را مطالعه کرده و در صورت عدم آشنایی، کد و کامنت‌های بخش‌هایی که کامل نشده‌اند را بررسی و تکمیل کنید.

۲. کلاس Model

این کلاس یک کلاس انتزاعی (abstract) است که مدل باید در چارچوب آن پیاده‌سازی شود. تمامی مدل‌های تعریف‌شده در ادامه باید از این کلاس مشتق شوند.

۳. ماژول Layer

در این بخش باید لایه‌های fully-connected را پیاده‌سازی کنید.
ورودی یک لایه متراکم (Dense Layer) در وزن‌های آن ضرب شده و سپس جمع می‌شود تا خروجی حاصل گردد. برای پیاده‌سازی این قسمت باید فایل linear.py را تکمیل نمایید.
شکل ارائه‌شده در فایل پروژه ایده کلی پیاده‌سازی این بخش را نشان می‌دهد.

۴. ماژول Activation

در این بخش باید توابع فعال‌سازی (Activation Functions) را پیاده‌سازی کنید. تابع stepبه عنوان نمونه پیاده‌سازی شده است، اما توابع زیر نیاز به تکمیل دارند:

- relu
- leaky relu
- sigmoid
- softmax

تابع tanhاختیاری و امتیازی است.

۵. ماژول Loss

در این بخش باید دو تابع هزینه زیر را پیاده‌سازی و کامل کنید:

- **MSE (Mean Squared Error)**
- **CE (Cross Entropy)**

۶. ماژول Optimizer

در این بخش باید بهینه‌سازها (Optimizers) را پیاده‌سازی نمایید.
در فصل اول درس با روش گرادیان کاهشی (Gradient Descent) آشنا شده‌اید، بنابراین لازم است فایل sgd.py را کامل کنید. در هر گام باید پارامترهای مدل را با الگوریتم به‌روزرسانی مناسب اصلاح نمایید.
پیاده‌سازی سایر بهینه‌سازها نظیر Adam نیز مجاز است.

نکات تکمیلی:

- مقدار Batch size و تعداد epoch باید به‌گونه‌ای تنظیم شوند که به بهبود نتیجه train و تست نهایی کمک کنند. در این قسمت حداقل ۷۰ درصد دقت train و ۶۰ درصد دقت test انتظار می‌رود.
- کدهای این فاز باید در یک نوت بوک با نام Phase2.ipynb در دسترس باشند. یک پوشه شامل پیاده‌سازی اولیه (کمکی) کدهای Mytorch در پروژه برای این فاز قرار گرفته است که قابل استفاده و تکمیل است.

فاز سوم (مقایسه توابع فعال‌سازی)

در این فاز اثر توابع فعال‌سازی مختلف بر کارایی MLP مورد بررسی است. دقت شود در این فاز نیز صرفا استفاده از کتابخانه MyTorch مجاز می‌باشد.

دامنه‌ی مقایسه:

- Sigmoid
- Tanh
- ReLU
- (alpha = 0.01) LeakyReLU

نکات تکمیلی:

- سایر مولفه‌ها (معماری، داده، بهینه‌ساز، تعداد epoch، seed و ...) باید ثابت نگه داشته شود.
- مقایسه این قسمت شامل یک جدول مقایسه و ۷ پلات (به ازای هر تابع فعال‌ساز) خواهد بود.
- ارائه تحلیل برمبنای نمودارها و جدول الزامی است.
- کدهای این فاز باید در یک نوت بوک با نام Phase3.ipynb در دسترس باشد. می‌توان از کدهای کمکی فاز ۲ نیز در این فاز کمک گرفت.

فاز چهارم (پیاده‌سازی شبکه KAN و مقایسه آن با MLP - امتیازی)

در خصوص شبکه KAN تحقیق و مطالعه نمایید و ضمن پیاده‌سازی آن (به کمک کتابخانه‌ها) نتایج آن را با MLP مقایسه نمایید.

نکات تکمیلی:

- مقایسه باید به صورت جدول و نمودار باشد.
- همچنین درخصوص تفاوت‌های ساختاری شبکه KAN و MLP در گزارش توضیح دهید.
- کدهای این فاز باید در یک نوت بوک با نام Phase4.ipynb در دسترس باشند.

لینک‌های مفید:

- https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
- https://arxiv.org/pdf/2404.19756

ضمیمه (شبکه عصبی KAN)

تقریبا اوایل اردیبهشت ۱۴۰۳ مقاله‌ای تحت عنوان Kolmogorov-Arnold Networks یا به اختصار KAN معرفی شد و سروصدای زیادی داشت. جدای سر و صدا و هیاهو حول مقاله، که راستی آزمایی آن با انجام تحقیقات و کارهای بیشتر در این حوزه امکان پذیر است، از این حیث که این مقاله نگاه جدیدی نسبت به آموزش در شبکه های عصبی داره قطعا ارزش بررسی رو داره. در این بخش از مجله بیشتر به

بررسی خود مقاله و شناخت نحوه عملکرد این گونه از شبکه‌ها و... می‌پردازیم. در ادامه ابتدا با یک تصویر کلی از مقاله روبرو خواهید شد، سپس به پیش نیازهای ریاضی برای درک دقیق این شبکه و در ادامه آن به جزئیات پیاده‌سازی آن می‌پردازیم. در آخر ضمن تحلیل توسعه‌پذیری و تفسیرپذیری این گونه از شبکه‌ها، نگاه مختصری به کارهای آتی مبتنی بر این مقاله داریم تا اگر علاقه‌مند به کسب اطلاعات بیشتر و عمیق‌تری درباره کاربردهای این گونه از شبکه‌ها هستید، بتوانید پا را یک پله فراتر بگذارید!

در ابتدای مقاله شکلی در قالب جدول آمده که تقریبا هر آنچه این مقاله قصد دارد به ما منتقل کند، در این شکل خلاصه شده و در قالب یک مقایسه بین MLP و KAN هاست که دربردارنده ایده کلی‌ای که این دو شبکه بر مبنای آن بنا شده‌اند و ... است. در این لحظه وارد جزئیات این شکل نخواهیم شد؛ صرفا این مورد رو به صورت انتزاعی در نظر داشته باشید که در شبکه‌های MLP ما تعدادی گره (نود) در قالب تعدادی لایه داشیم که به هم متصل بودند و روی یال‌ها ما وزن‌هایی داشتیم که قابل یادگیری بود و در نودها صرفا عملیات جمع ساده و اعمال یک تابع فعال‌ساز از پیش تعیین شده وجود داشت و این غیرخطی‌سازی به صورت یک تابع از پیش معین و غیرقابل تغییر وجود داشت اما در KAN ها به جای اینکه روی یال‌ها وزن‌هایی قابل آموزش وجود داشته باشد، توابعی روی هر کدام از گره‌ها وجود دارد که قابل یادگیری هستند و ما در یال‌ها عملگر جمع ساده داریم! به بیانی دیگر ما این بار خود توابع را مستقیما یاد میگیریم. در واقع MLP ها از تئوری تقریب همگانی و KAN ها از تئوری نمایش کولگومورف آرنولد تبعیت می‌کنند. در ادامه به بررسی اینکه هر کدام بیانگر چه چیزی هستند مفصلاً می‌پردازیم.

Model	**Multi-Layer Perceptron (MLP)**	**Kolmogorov-Arnold Network (KAN)**
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) *fixed* activation functions on *nodes*; *learnable* weights on *edges*	(b) *learnable* activation functions on *edges*; sum operation on *nodes*
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) $\mathrm{MLP}(\mathbf{x})$; $\mathbf{W}_3$, σ_2 *nonlinear, fixed*, $\mathbf{W}_2$, σ_1, $\mathbf{W}_1$ *linear, learnable*; $\mathbf{x}$	(d) $\mathrm{KAN}(\mathbf{x})$; $\mathbf{\Phi}_3$, $\mathbf{\Phi}_2$ *nonlinear, learnable*, $\mathbf{\Phi}_1$; $\mathbf{x}$

In MLP, we assign a fixed set of activation functions to the nodes. However, the idea of KAN is that instead of placing activation functions on the nodes, we place them on the edges of the network! In essence, each edge has a learnable activation function, and the results of the edges at each node are simply summed together.

خب حالا که یک دید کلی از ماجرا پیدا کردیم، باید به بررسی مفصل مواردی که معرفی کردیم بپردازیم. ابتدا به سراغ تئوری تقریب همگانی (Universal Approximation theorem) می‌رویم. قضیه تقریب همگانی بیان می‌کند که یک شبکه عصبی پیش‌خور با حداقل یک لایه مخفی و تعداد محدودی نورون می‌تواند هر تابع پیوسته‌ای را با دقت دلخواه تقریب بزند، مشروط بر اینکه وزن‌ها و توابع فعال‌سازی مناسب انتخاب شوند. این قضیه پایه نظری قدرت و انعطاف‌پذیری شبکه‌های عصبی در مدل‌سازی روابط و الگوهای پیچیده در داده‌ها را فراهم می‌کند. این اطمینان را می‌دهد که با شبکه‌ای به اندازه کافی بزرگ و آموزش مناسب، می‌توان از شبکه‌های عصبی برای انجام طیف گسترده‌ای از وظایف مانند رگرسیون، طبقه‌بندی و نگاشت‌های پیچیده‌تر استفاده کرد. حال به سراغ تئوری نمایش کولموگروف-آرنولد (Kolmogorov-Arnold representation theorem) می‌رویم. در قلب این مفهوم بنیادین، تئوری ریاضی‌ای قرار دارد که توسط ولادیمیر آرنولد و آندری کولموگروف توسعه یافته است. این تئوری بیان می‌کند که توابع چندمتغیره پیچیده را می‌توان به توابع ساده‌تر یک‌بعدی تجزیه کرد، که این پایه و اساس ساختار منحصر به فرد KAN ها را فراهم می‌آورد.

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right).$$

where $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$.

پرسشی که طبیعتاً برایتان پیش آمده این است که این "توابع ساده‌تر یک‌بعدی" چه هستند؟ برای هر کسی که کمی با ریاضیات یا گرافیک محاسباتی آشنا باشد، این توابع همان چندجمله‌ای‌های تکه‌ای هستند که به نام اسپلاین‌ها (spline) شناخته می‌شوند. اسپلاین‌ها توابع ریاضی‌ای هستند که امکان ایجاد منحنی‌های هموار با اتصال مجموعه‌ای از نقاط کنترل را فراهم می‌کنند. اسپلاین‌ها، انعطاف‌پذیری در تنظیم شکل منحنی را ارائه می‌دهند، این در حالی است که پیوستگی و همواری بین قطعات مجاور را نیز تضمین می‌کنند. برای ایجاد یک اسپلاین، معمولاً با مجموعه‌ای از نقاط کنترل که مسیر منحنی را تعریف می‌کنند، شروع می‌شود. سپس منحنی با درون‌یابی یا تقریب مسیر بین این نقاط کنترل با استفاده از توابع پایه، مانند B-اسپلاین‌ها یا منحنی‌های بزیه (bezier curves)، ساخته می‌شود.به طور خلاصه، اسپلاین‌ها ابزار متنوعی برای نمایش منحنی‌ها یا سطوح پیچیده با دقت و انعطاف‌پذیری فراهم می‌کنند که همین موضوع، آنها را در زمینه‌های مختلف ارزشمند می‌سازد. در اینجا وارد ریاضی و نحوه اثبات این موضوع نمی‌شویم تخمین

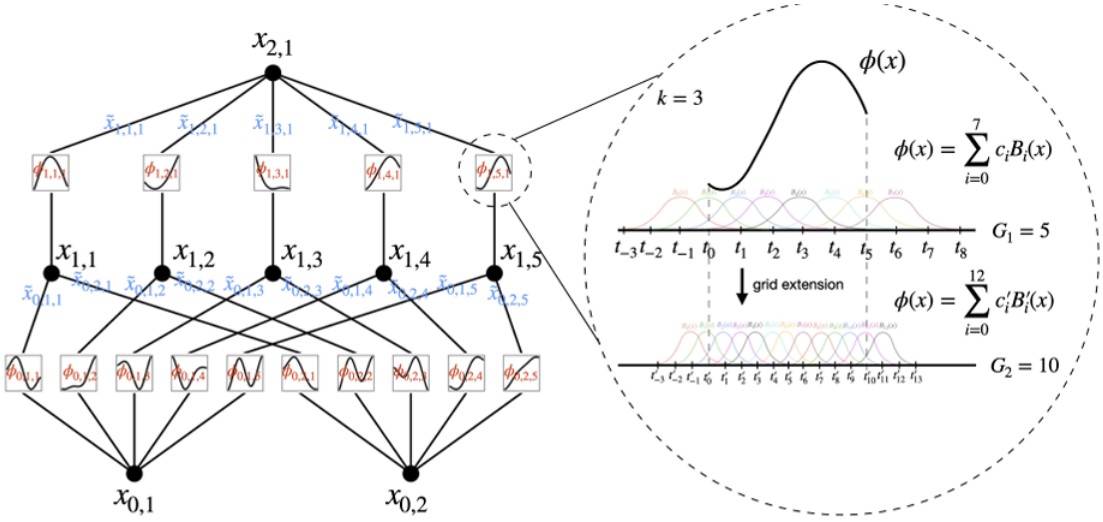
هر منحنی فقط و فقط به تعداد نقاط کنترلی و خود این نقاط وابسته است و توابع پایه برای تعداد معینی نقاط کنترلی ثابت و غیر متغیر است.

اما اسپلاین‌ها چگونه در معماری KAN استفاده و بهره‌برداری می‌شوند؟ برای پاسخ به این سوال باید به سراغ هر کدام از گره‌های شبکه کولموگروف-آرنولد برویم. در هر کدام از گره‌ها یک تابع قابل یادگیری داریم که از دو بخش قابل یادگیری و یک بخش ثابت تشکیل شده‌اند که هر یک از آنها در یک وزن ثابت کنترلی ضرب شده و با یکدیگر جمع می‌شوند. همانطور که پایین‌تر قابل مشاهده است، تابع ثابت b(x) همان بخش ثابت (تابع سیلو) و spline(x) همان بخش قابل یادگیری است که برابر مجموع حاصل ضرب نقاط کنترلی در توابع پایه متناظر آنها است. به بیانی دیگر ما باید برای تخمین توابع مجموعه‌ای از نقاط کنترلی خوب را حدس بزنیم که تعداد این نقاط به عنوان یک هایپر پارامتر برای مسئله ماست(grid size).

$$\phi(x) = w_b b(x) + w_s \text{spline}(x).$$

$$b(x) = \text{silu}(x) = x/(1 + e^{-x})$$

$$\text{spline}(x) = \sum_i c_i B_i(x)$$



با استناد و تکیه به این ساختار می‌توان یک شبکه KAN مشابه نسبت یک MLP استاندارد با عملکرد یکسان، حدود ۱۰۰ برابر پارامتر های کمتر و ۱۰ برابر کند تر در بخش آموزش است. به عبارتی دیگر ۱۰ برابر بهتر از یک شبکه پرسپترون چندلایه عمل می‌کند که به همین دلیل انتظار می‌رود از قدرت توسعه‌پذیری بهتری برخوردار باشد. همچنین MLP ها مانند یک جعبه‌ی سیاه و نامعلوم می‌مانند و تفسیر وزن های شبکه و دریافتن ارتباطات و الگوها از آن بسیار دشوار و ناممکن است. اما در KAN ها به دلیل اینکه ما مستقیما توابع و یا به بیانی ساده‌تر ارتباطات را یاد می‌گیریم؛ می‌توانیم از تفسیرپذیری بالاتری برخوردار باشیم. به طور مثال و برای درک بهتر اگر مجموعه دادگان از تابع زیر تولید شده باشند؛ انتظار می‌رود KAN ها مطابق روند زیر به کشف این موضوع و با تفسیرپذیری بهتری این موضوع را یاد بگیرند.

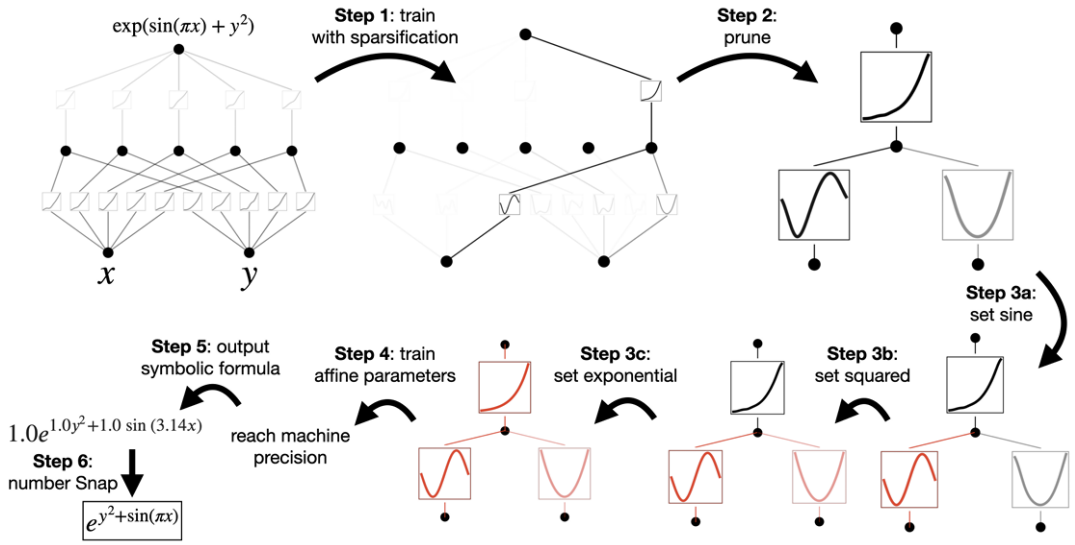


Figure 2.4: An example of how to do symbolic regression with KAN.

از زمان انتشار این مقاله تاکنون کارهای متفاوتی پیرامون آن اتفاق افتاده که می‌توان به شبکه های کانولوشنی کولموگروف-آرنولد و ... اشاره کرد که اکثر و اغلب این کارها در مخزن گیتی با نشانی github.com/mintisan/awesome-kan گردآوری و تجمیع شده است.

منبع عکس ها: arxiv.org/abs/2404.19756