



SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET PRIMIJENJENE MATEMATIKE I INFORMATIKE

Jednostavna implementacija Kolmogorov-Arnold mreže

SEMINARSKI RAD IZ KOLEGIJA
OBRADA PRIRODNOG JEZIKA METODAMA DUBINSKOG UČENJA

Student:

Marin Kovač

Osijek, 2024.

1 | Uvod

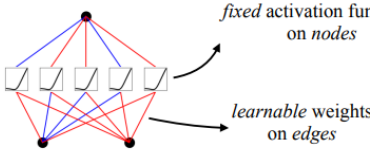
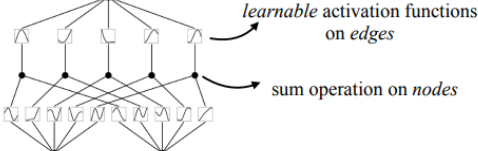
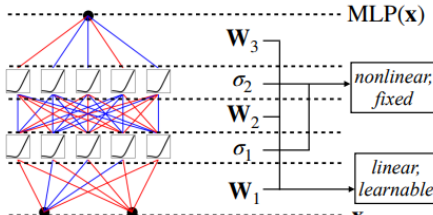
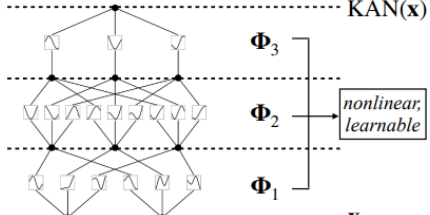
Seminar, odnosno završni projekt, dio je nastavnog procesa kolegija "Obrada prirodnog jezika metodama dubinskog učenja".

Cilj projekta je razumjevanje i implementacija Kolmogorov Arnold mreže. Implementacija prati originalni papir "KAN: Kolmogorov-Arnold Networks"[1]. Rad sadrži osnovne ideje originalnog papira i prolazi kroz jednostavnu pytorch implementaciju.

Kod projekta nalazi se na <https://github.com/KovaCro/OPJTDU-projekt>

2 | Kolmogorov-Arnold mreža

Duboke neuronske mreže su dovele do velikog razvoja umjetne inteligencije u zadnjem desetljeću. Iako su se pokazale kao dobro rješenje, pate od problema skaliranja, interpretabilnosti, podatkovne neefikasnosti i ostalog. Kolmogorov Arnold mreže pojavile su se kao alternativa standardnim MLP-ovima. Za razliku od MLP-ova, KAN umjesto težina na bridovima ima funkcije jedne varijable parametrizirane kao splajn i nema aktivacijskih funkcija na čvorovima.

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a)  fixed activation functions on nodes learnable weights on edges	(b)  learnable activation functions on edges sum operation on nodes
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	(c)  MLP(x) \mathbf{W}_3 σ_2 \mathbf{W}_2 σ_1 \mathbf{W}_1 \mathbf{x} nonlinear, fixed linear, learnable	(d)  KAN(x) Φ_3 Φ_2 Φ_1 \mathbf{x} nonlinear, learnable

Slika 2.1: Slika iz originalnog papira [1]

2.1 Motivacija i arhitektura

Slično MLP-ovima koji su motivirani teoremom o univerzalnoj aproksimaciji, tako motivacija za KAN mreže proizlazi iz Kolmogorov Arnoldovog teorema reprezentacije.

Teorem 1 (Kolmogorov Arnold teorem reprezentacije). *Neka je n prirodan broj. Za $n \geq 2$ postoje neprekidne realne funkcije $\phi_{p,q}(x)$ na segmentu $E^1 = [0, 1]$ takve da se*

svaka realna funkcija $f(x_1, \dots, x_n)$ na n -dimenzionalnoj jediničnoj kocki E^n može prikazati u obliku

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left[\sum_n^{p=1} \phi_{p,q}(x_p) \right] \quad (2.1)$$

, gdje su $\Phi_q(y)$ neprekidne realne funkcije.

Kako bi nam gornji teorem bio koristan u strojnom učenju, moramo ga generalizirati u slojeve i proizvoljnu širinu. To je upravo ono što su originalni autori napravili. Definirali su KAN sloj $\mathcal{K}_{m,n}$ gdje je n ulazna dimenzija i m izlazna dimenzija kao parametriziranu matricu funkcija jedne varijable $\Phi = \{\Phi_{i,j}\}_{i \in [m], j \in [n]}$, odnosno

$$\mathcal{K}_{m,n}(x) = \Phi x \quad \text{gdje} \quad \forall i \in [m], (\Phi x)_i = \sum_{j=1}^n \phi_{i,j}(x_j)$$

Iz toga se lagano vidi da se jednadžba 2.1 može zapisati u obliku:

$$f(x_1, \dots, x_n) = \mathcal{K}_{1,2n+1} \mathcal{K}_{2n+1,n}(x_1, \dots, x_n)$$

Slično kao kod MLP-ova, Kolmogorov Arnold mrežu dobijemo slaganjem KAN slojeva $\mathcal{K}_{m,n}$. Trenutno nema čvrstih matematičkih dokaza da je familija KAN mreža (osim $\mathcal{K}_{1,2n+1} \mathcal{K}_{2n+1,n}$) gusta u prostoru funkcija. Ono što su autori uspjeli pokazati je postojanje KAN-a koji koristi B-splajnove i može aproksimirati funkciju (više u originalnom papiru [1]) s određenom greškom.

2.2 B-splajnovi

Kako bi parametrizirali funkciju, moramo definirati neku vrstu bazne funkcije koja koristi koeficijente. "Učenje" funkcije svodi se na "učenje" koeficijenata. Kolmogorov teorem ne postavlja nikakve uvijete na odabir funkcija ϕ . Po Stone-Weierstrass teoremu, svaka neprekidna funkcija na segmentu može se aproksimirati polinomom. Autori su za familiju funkcija odabrali B-splajnove koji su generalizacija splajn funkcija (po djelovima polinomi koji imaju dodatna svojstva, ali to izlazi iz okvira ovog rada). B-splajnovi su funkcije oblika:

$$B(x) = \sum_{i=1}^G c_i B_{i,k}(x)$$

, gdje je k red splajna, a G broj čvorova. U papiru dodatno augmentiraju broj čvorova s $2k$ čvorova jer ponavljanje čvorova na krajevima k puta dovodi krajnje točke u poravnanje s kontrolnim poligonom. (Više o splajnovima u [2])

2.3 Implementacija

Mrežu smo implementirali u pytorchu. Pytorch ima mogućnost automatskog računanja gradijenta što nam uvelike olakšava implementaciju jer ne moramo pisati

algoritam povratne propagacije prilikom razvijanja modela. Implementacija i inicijalizacija KAN mreže ista je kao i kod ostalih mreža. Konstruktor prima broj čvorova po sloju i parametre splajnova (broj čvorova, red splajna, raspon grida). Pri inicijalizaciji prođe kroz matricu slojeva i "zakači" slojeve na mrežu.

```

1 class KAN(torch.nn.Module):
2     def __init__(
3         self,
4         layers,
5         grid_size=5,
6         grid_range=[-1, 1],
7         spline_order=3,
8     ):
9         super(KAN, self).__init__()
10        self.grid_size = grid_size
11        self.spline_order = spline_order
12        self.layers = torch.nn.ModuleList()
13        for in_features, out_features in zip(layers, layers[1:]):
14            self.layers.append(
15                KANLinear(
16                    in_features,
17                    out_features,
18                    grid_size=grid_size,
19                    spline_order=spline_order,
20                    grid_range=grid_range,
21                )
22            )

```

Uočimo da zbog jednostavnosti svaki sloj ima iste splajn parametre. U klasi mreže nam još fali forward funkcija koja je univerzalna za sve tipove mreže.

```

1     def forward(self, x):
2         for layer in self.layers:
3             x = layer(x)
4         return x

```

Sljedeće je KAN sloj koji prima dimenzije ulaza i izlaza te parametre splajna. Pri inicijalizaciji generiramo uniformni grid augmentiran s $2k + 1$ točkama kao što je ranije spomenuto te ga u memoriju kao konstantu. Uz to definiramo težine za baznu funkciju i koeficijente splajnova kao parametre modela.

```

1 class KANLinear(torch.nn.Module):
2     def __init__(
3         self,
4         in_features,
5         out_features,
6         grid_size=5,
7         grid_range=[-1, 1],
8         spline_order=3,
9     ):
10        super(KANLinear, self).__init__()

```

```

11     self.in_features = in_features
12     self.out_features = out_features
13     self.grid_size = grid_size
14     self.spline_order = spline_order
15     h = (grid_range[1] - grid_range[0]) / grid_size
16     grid = (torch.arange(-spline_order, grid_size +
spline_order + 1) * h + grid_range[0]).expand(in_features, -1).
contiguous()
17     self.register_buffer("grid", grid)
18     self.base_weight = torch.nn.Parameter(torch.Tensor(
out_features, in_features))
19     self.spline_weight = torch.nn.Parameter(torch.Tensor(
out_features, in_features, grid_size + spline_order))

```

Sada trebamo implementirati propagaciju unaprijed. U papiru[1] implementacija je opisana sljedećim formulama:

$$\phi(x) = w_b b(x) + w_s \text{spline}(x) \quad (2.2)$$

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

te je dodatno predloženo da se $\text{spline}(x)$ parametrizira kao linearna kombinacija B-splajnova

$$\text{spline}(x) = \sum_i c_i B_i(x) \quad (2.3)$$

Uočimo da kada imamo splajn oblika 2.3 gdje su c_i i w_s parametrizirani, možemo zanemariti w_s . Iz toga nam slijedi implementacija propagacije unaprijed.

```

1     def forward(self, x):
2         tmp = x.shape
3         x = x.reshape(-1, self.in_features)
4         base_activation = torch.nn.SiLU()
5         base_output = torch.nn.functional.linear(base_activation(x)
, self.base_weight)
6         spline_output = torch.nn.functional.linear(
7             self.splines(x).view(x.size(0), -1),
8             self.spline_weight.view(self.out_features, -1),
9         )
10        output = base_output + spline_output
11        output = output.reshape(*tmp[:-1], self.out_features)
12        return output

```

Jedino što nam je preostalo jest implementacija funkcije splines koja nam vraća baze splajnova.

Za izračun B-splajnova možemo koristiti Cox-de Boor rekurzivnu formulu koja glasi

$$B_{i,0}(t) := \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & \text{inace} \end{cases}$$

$$B_{i,p}(t) := \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t)$$

gdje je p red polinoma $B_{i,p}(t)$. Kako bi izbjegli rekurzivne pozive i računali u tenzorskoj notaciji koristit ćemo dinamičko programiranje. Riješenje u reduciranom obliku izgleda ovako;

```

1     def splines(self, x):
2         grid = self.grid
3         x = x.unsqueeze(-1)
4         bases = ((x >= grid[:, :-1]) & (x < grid[:, 1:])).to(x.
dtype)
5         for p in range(1, self.spline_order + 1):
6             bases = ((x - grid[:, : -(p + 1)])/(grid[:, p:-1] -
grid[:, : -(p + 1)])*bases[:, :, :-1]) + ((grid[:, p + 1 :] - x)
/(grid[:, p + 1 :] - grid[:, 1:-p]))* bases[:, :, 1:])
7         return bases.contiguous()

```

2.4 Testiranje

Ispitali smo model na jednostavnom problemu kako bi bili sigurni da je kod ispravno razvijen. Testirali smo KAN na problemu POS taganja. Za podatke smo uzeli treebank bazu te smo za ugrađivanje vektora uzeli predtrenirane vektore "wiki-news-300d-1M". Podatke smo predprocesirali klizajućim prozorom veličine 3 i metodom konkatencije. Za slojeve KAN mreže uzeli smo slojeve veličine [ulazna dimenzija, 64, izlazna dimenzija]. Uz veličinu batcha 128 i 10 epoha treninga naš model je postigao preciznost $> 95\%$.

Jupyter bilježnica u kojoj se nalazi test se nalazi na [githubu](#).

Literatura

- [1] Z. ZIMING LIU, YIXUAN WANG, SACHIN VAIDYA, FABIAN RUEHLE, JAMES HAL-
VERSION, MARIN SOLJAČIĆ, THOMAS Y. HOU, MAX TEGMARK, *KAN: Kolmogorov-
Arnold Networks*, arXiv:2404.19756
- [2] LES PIEGL , WAYNE TILLER, *The NURBS Book*, Part of the book series: Mono-
graphs in Visual Communication