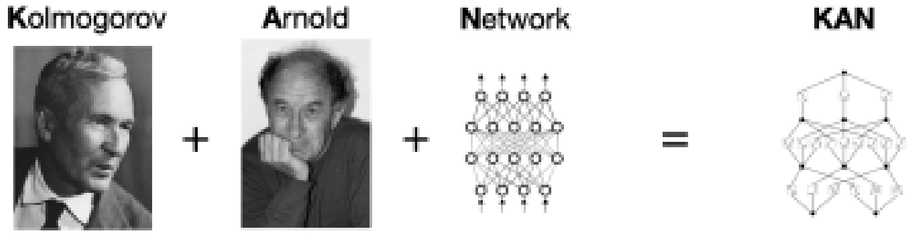
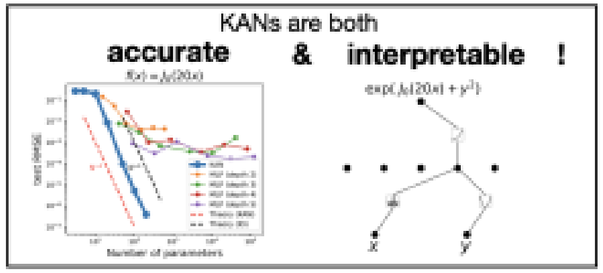
KAN: KOLMOGOROV-ARNOLD NETWORKS

Comparison with MLPs to evaluate model efficiency

What Are Kolmogorov-Arnold Networks (KANs)?

proposed Kolmogorov-Arnold networks are in honor of two great late mathematicians, Andrey Kolmogorov and Vladimir Arnold. KANs are mathematically sound, accurate and interpretable.





Kolmogorov - Arnold Theorem

(inspiration of KAN model)

1. Pass input features through uni-variate functions $\boldsymbol{\phi_1}(X_1)\ \ \boldsymbol{\phi_2}(X_2)\ \ \ \boldsymbol{\phi_3}(X_3)\ ....\ \boldsymbol{\phi_n}(X_n)$

2. Sum these up $\boldsymbol{\phi_1}(X_1) + \boldsymbol{\phi_2}(X_2) + \boldsymbol{\phi_3}(X_3)\ ....\ + \boldsymbol{\phi_n}(X_n) = \sum_{q=1}^{n} \phi_q(X_q)$

3. Pass the sum through another function $\boldsymbol{f}\left(\sum_{q=1}^{n} \phi_q(X_q)\right)$

3. Repeat M times $\sum_{p=1}^{m} \boldsymbol{f}_p\left(\sum_{q=1}^{n} \phi_{pq}(X_q)\right)$

Kolmogorov-Arnold representation theorem states that if f is a multivariate continuous function on a bounded domain, then it can be written as a finite composition of continuous functions of a single variable and the binary operation of addition. More specifically, for a smooth $f : [0,1]^n \to \mathbb{R}$,

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

Example of Kolmogorov-Arnold Theorem

Classic house pricing dataset

# Bedrooms	Area (Sq Ft)	Age	Price
2	18	25	**1000**
1	6	4	**500**
2	12	35	**1200**

Instead of writing in the "multi-variate" form:

Price = F(*Bedrooms, Area, Age*)

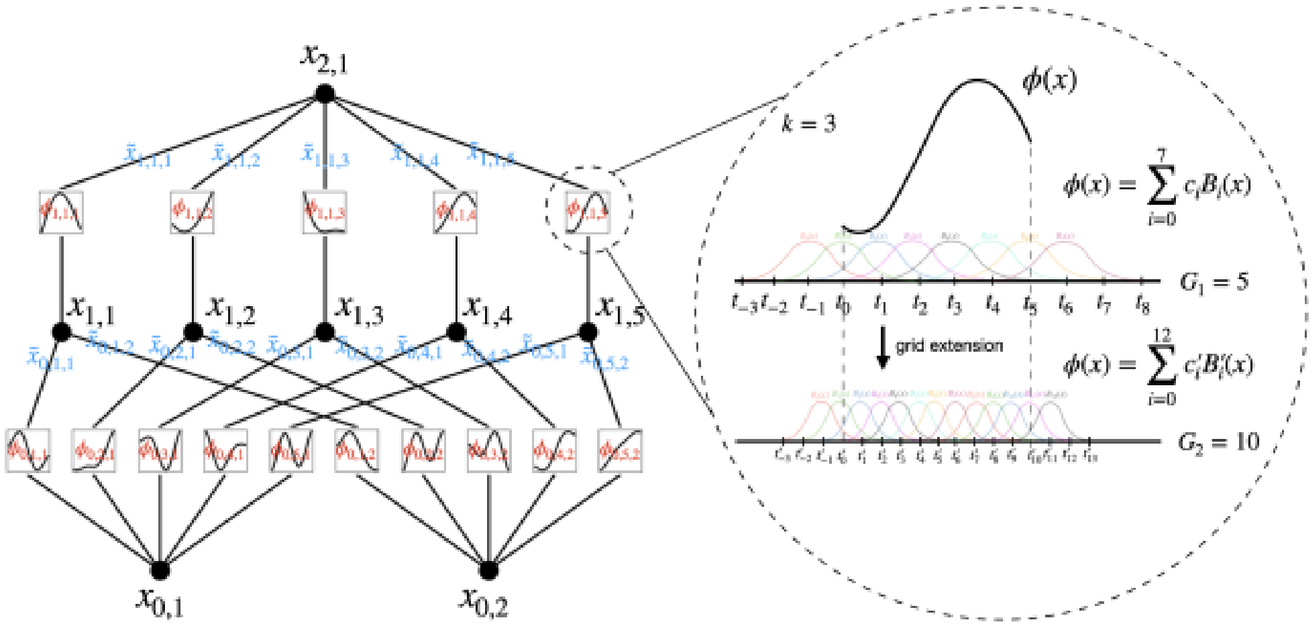
We can write:

$$\text{Price} = \boldsymbol{f}(\boldsymbol{\phi}_{B}(\text{Bedrooms}) + \boldsymbol{\phi}_{Sq}(\text{Area}) + \boldsymbol{\phi}_{A}(\text{Age}))$$

Or more generally:

$$\begin{aligned}\text{Price} = {} & \boldsymbol{f}_1(\boldsymbol{\phi}_{B1}(\text{Bedrooms}) + \boldsymbol{\phi}_{Sq1}(\text{Area}) + \boldsymbol{\phi}_{A1}(\text{Age})) + \\ & \boldsymbol{f}_2(\boldsymbol{\phi}_{B2}(\text{Bedrooms}) + \boldsymbol{\phi}_{Sq2}(\text{Area}) + \boldsymbol{\phi}_{A2}(\text{Age})) + \\ & \boldsymbol{f}_3(\boldsymbol{\phi}_{B3}(\text{Bedrooms}) + \boldsymbol{\phi}_{Sq3}(\text{Area}) + \boldsymbol{\phi}_{A3}(\text{Age})) + \\ & \ldots. \\ & \boldsymbol{f}_m(\boldsymbol{\phi}_{B\,m}(\text{Bedrooms}) + \boldsymbol{\phi}_{Sq\,m}(\text{Area}) + \boldsymbol{\phi}_{A\,m}(\text{Age}))\end{aligned}$$

Structure of Kolmogorov-Arnold Networks (KANs)

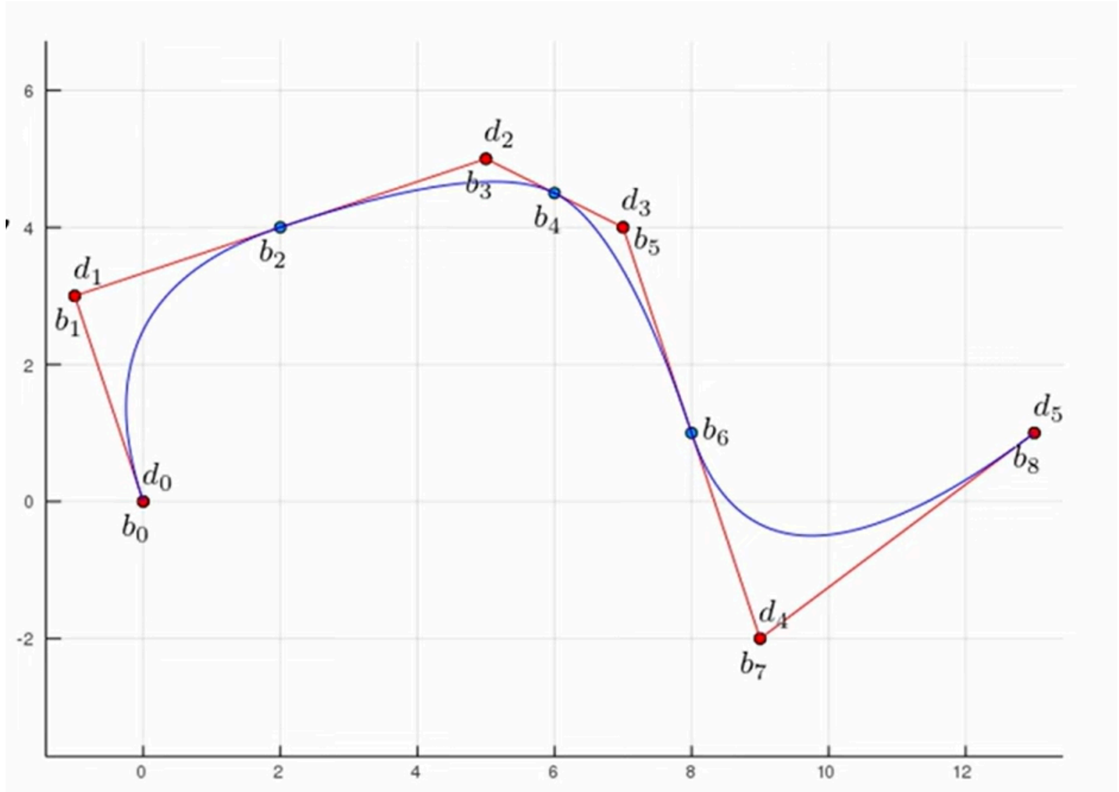


Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.

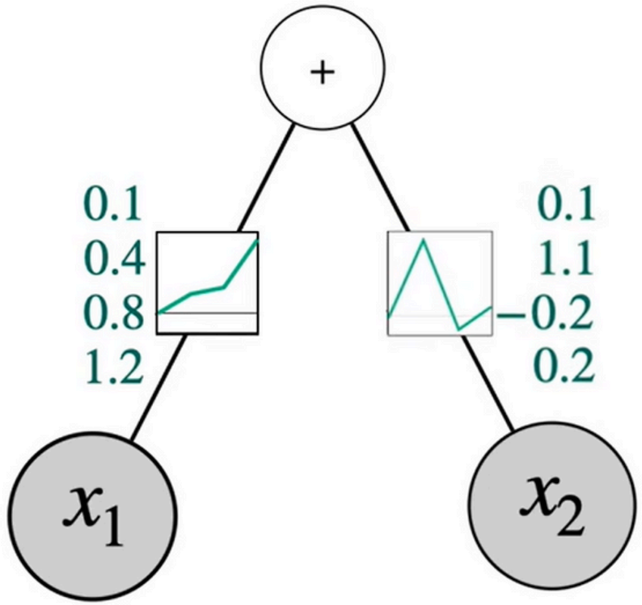
B splines: A control grid

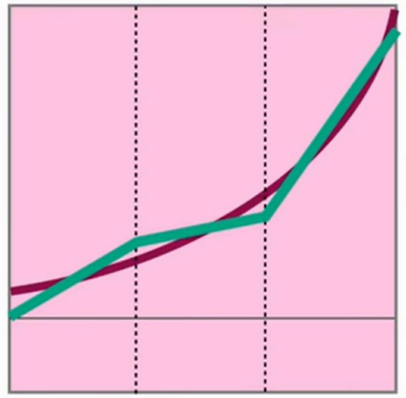
- B spline is the **piecewise polynomial curves** composed of a sequence of lower order polynomial segments

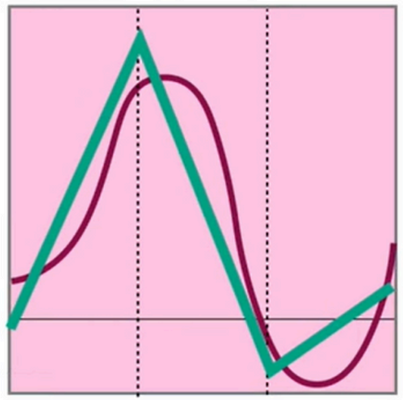
- B spline also have **local control**, that means changing the control points only will change the local area of the curve.

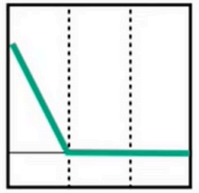


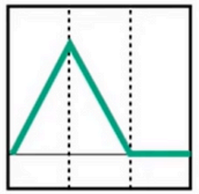
Linear B-Splines

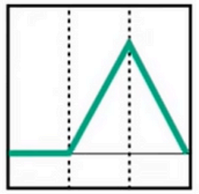




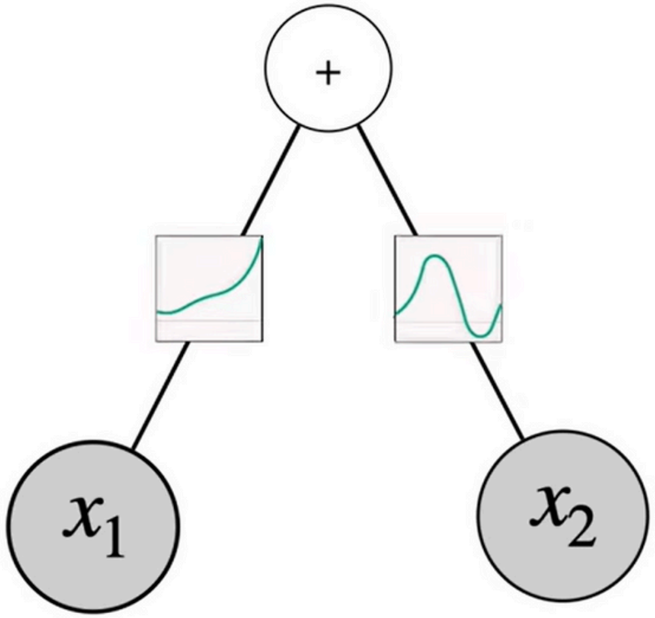


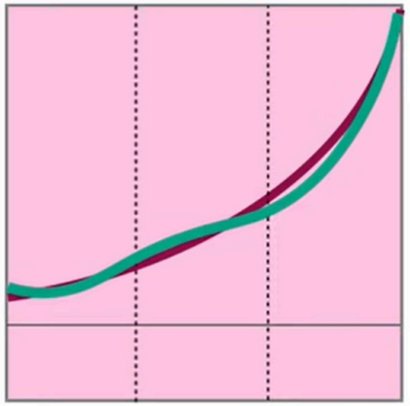


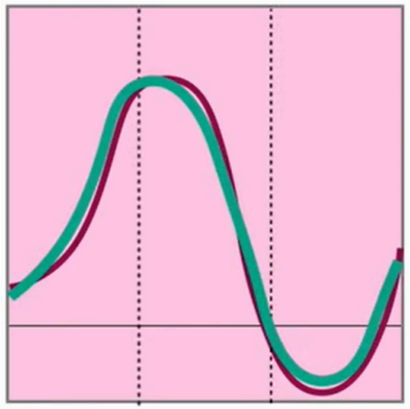


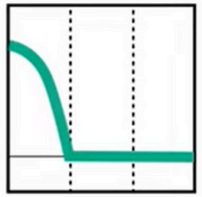


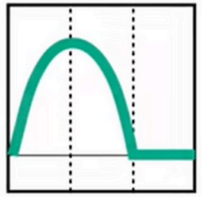
Quadratic B-Splines

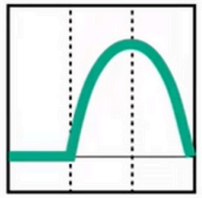


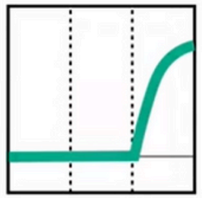




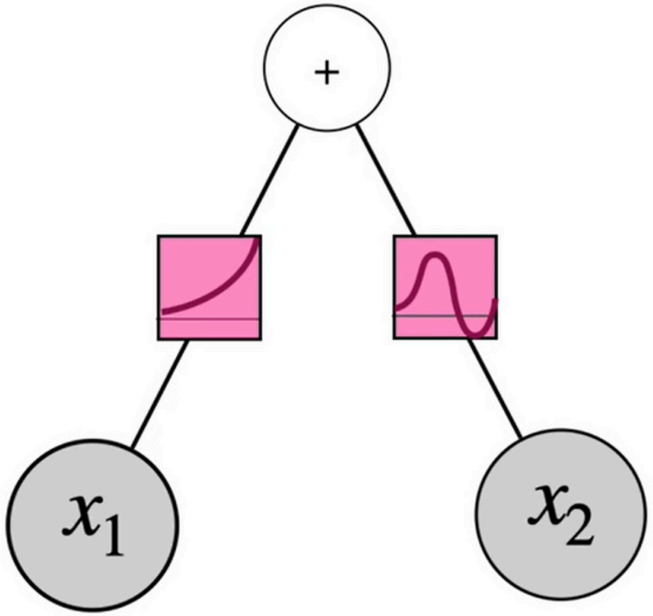


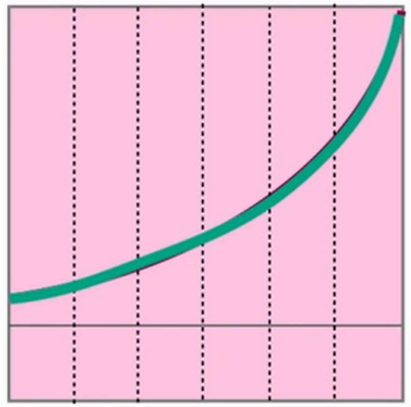


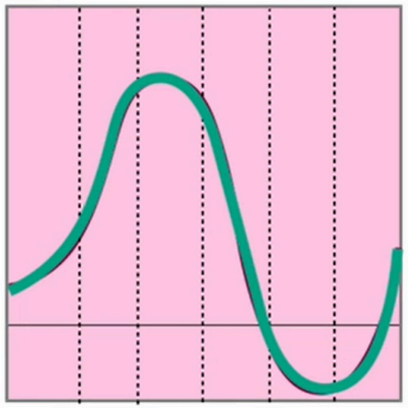


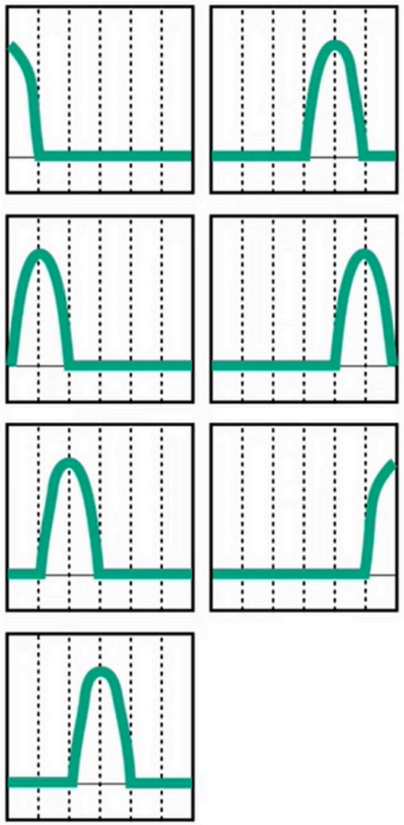


More B-Splines

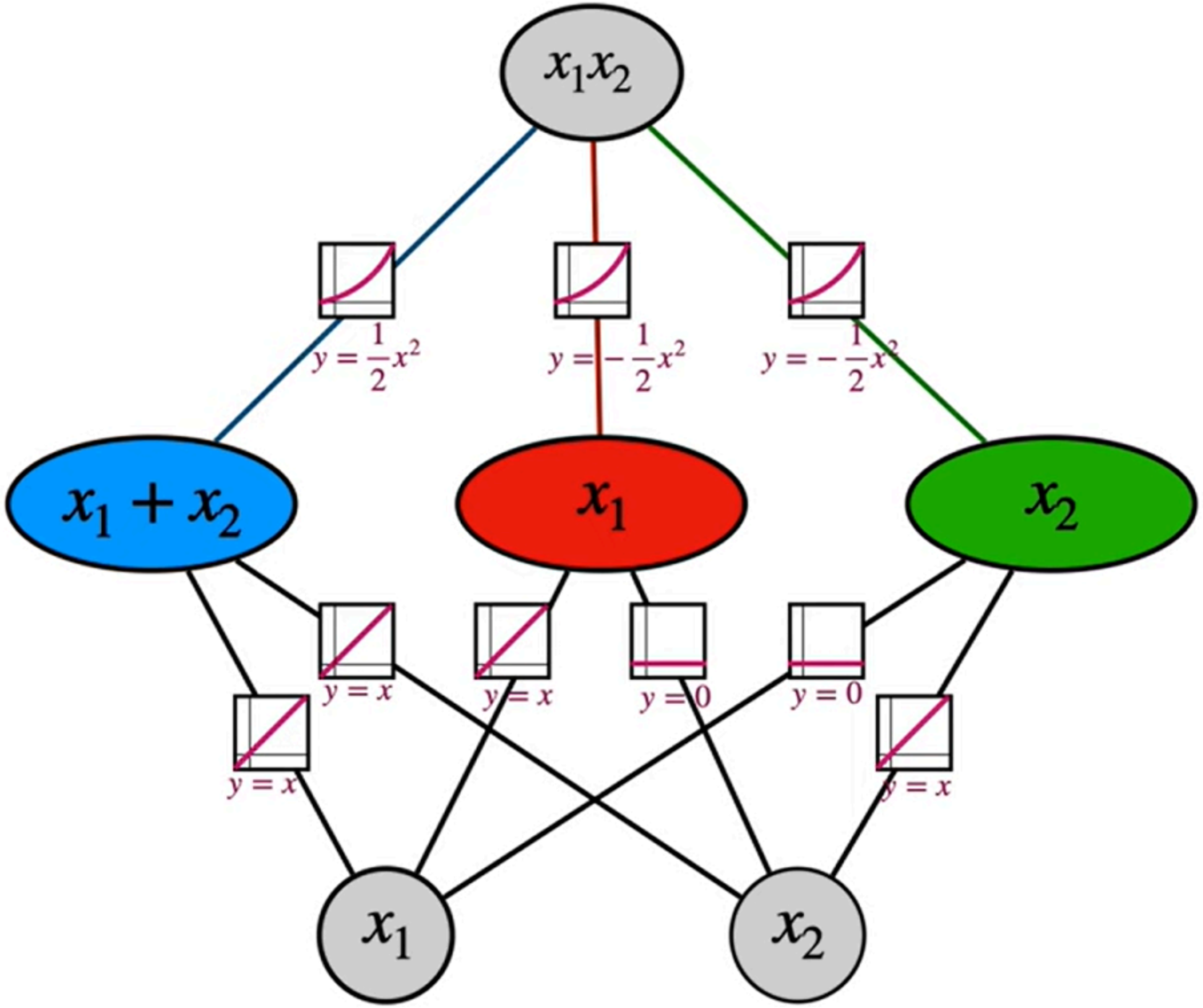




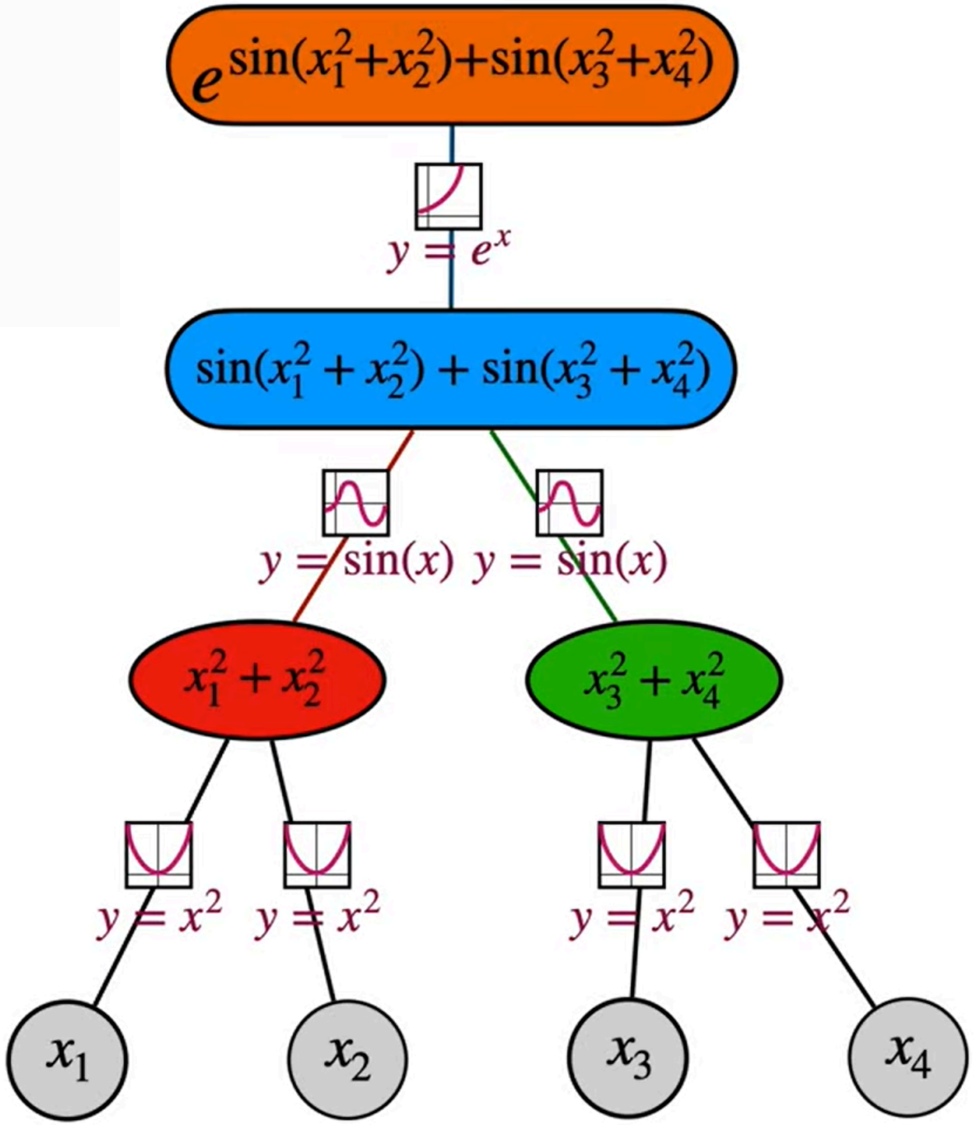




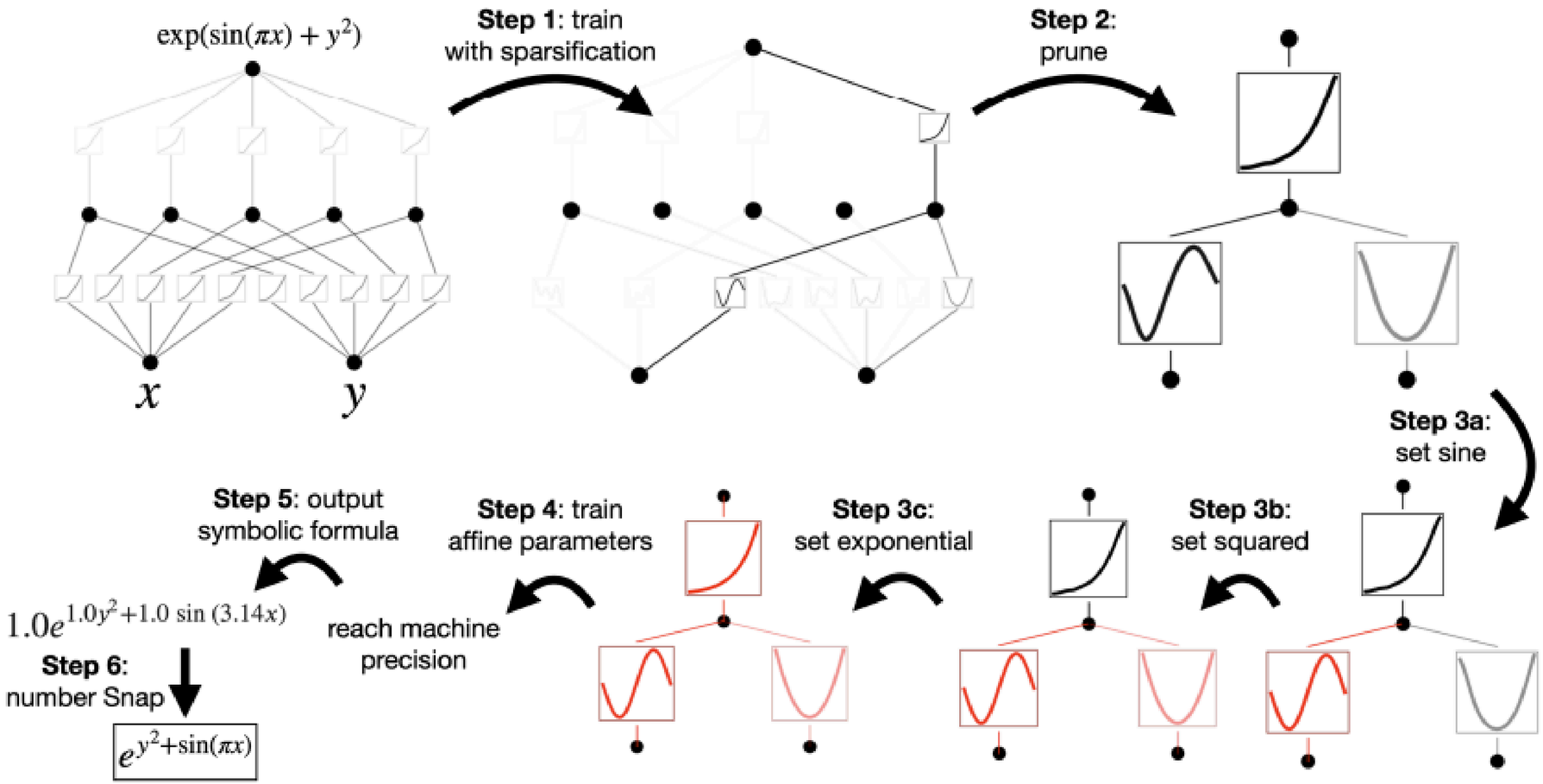
Example of functional steps of KAN



Example of functional steps of KAN

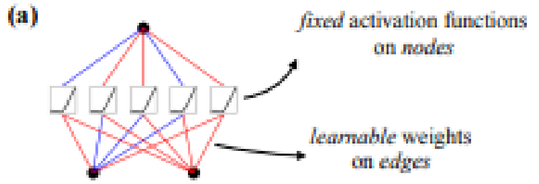
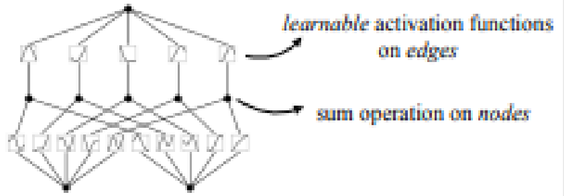
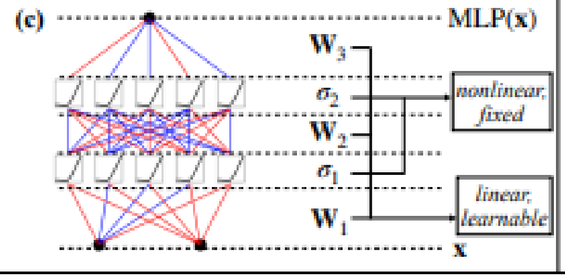
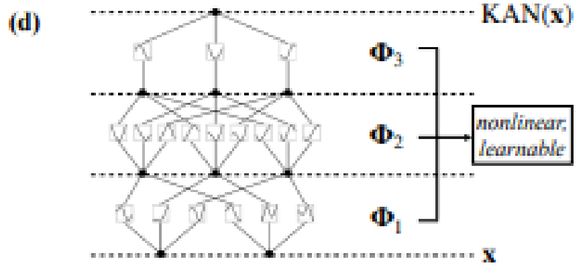


Simplification techniques of KAN



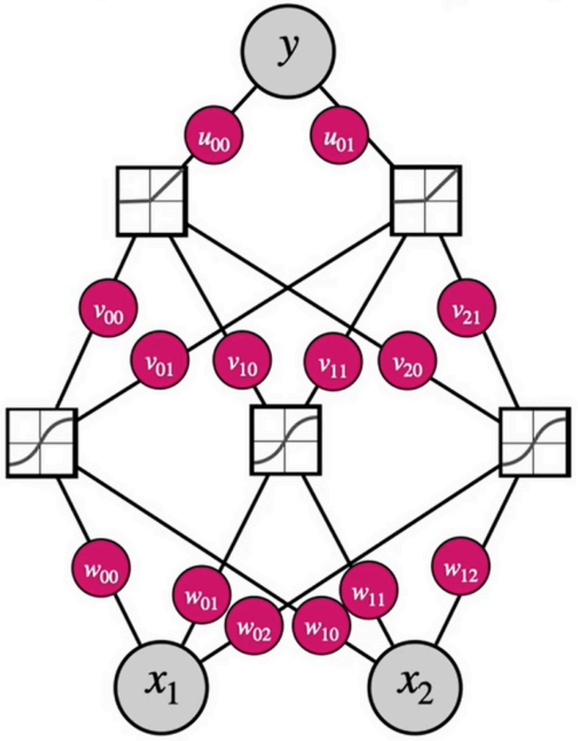
(An example of how to do symbolic regression with KAN.)

Kolmogorov-Arnold Networks (KANs) vs. Multi-Layer Perceptrons (MLPs)

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)		
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)		

Kolmogorov-Arnold Networks (KANs) vs. Multi-Layer Perceptrons (MLPs)

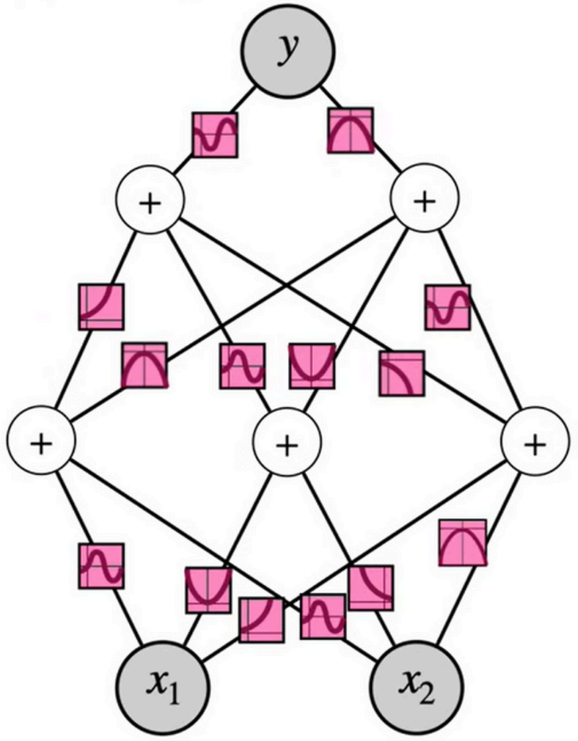
MLP (Multi-Layer Perceptron)



Fixed activation functions

Train weights

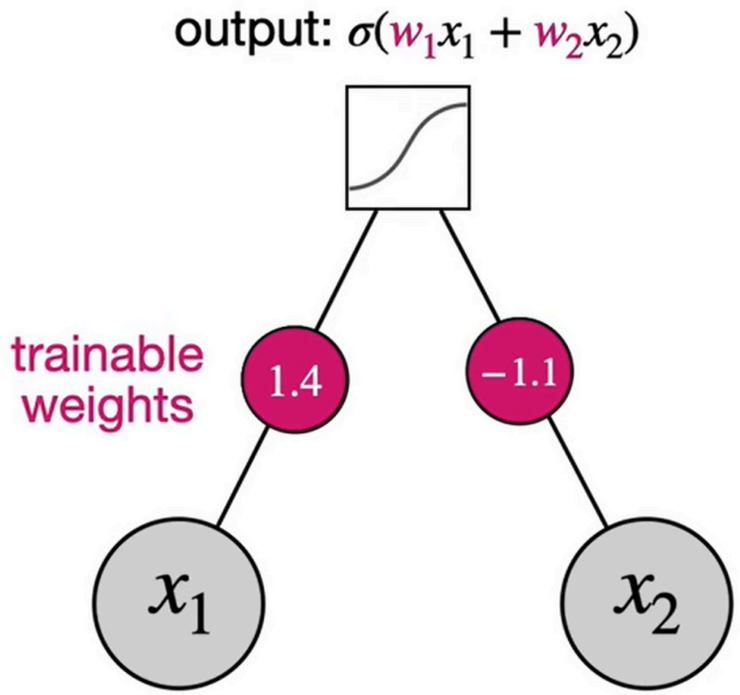
KAN (Kolmogorov-Arnold Network)

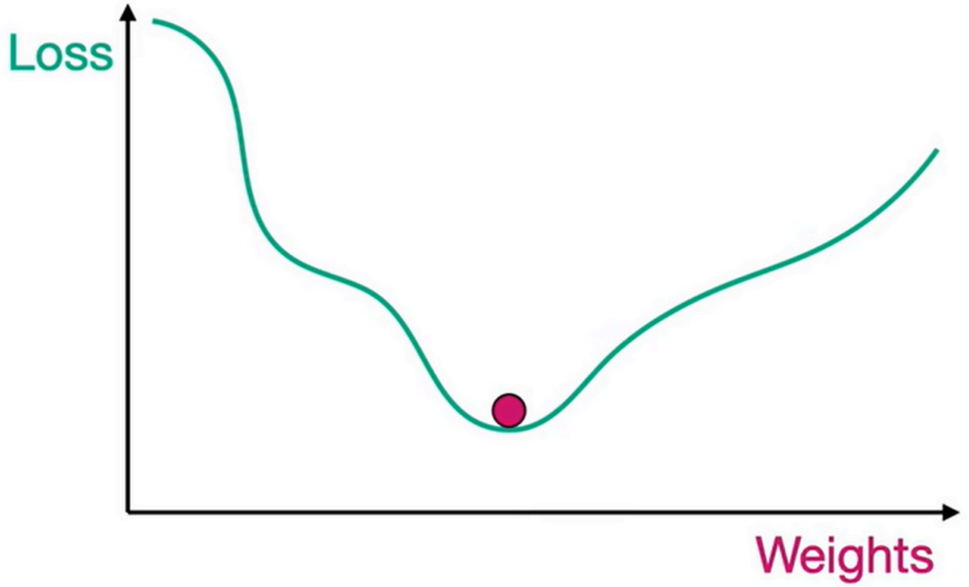


Fixed weights

Train activation functions

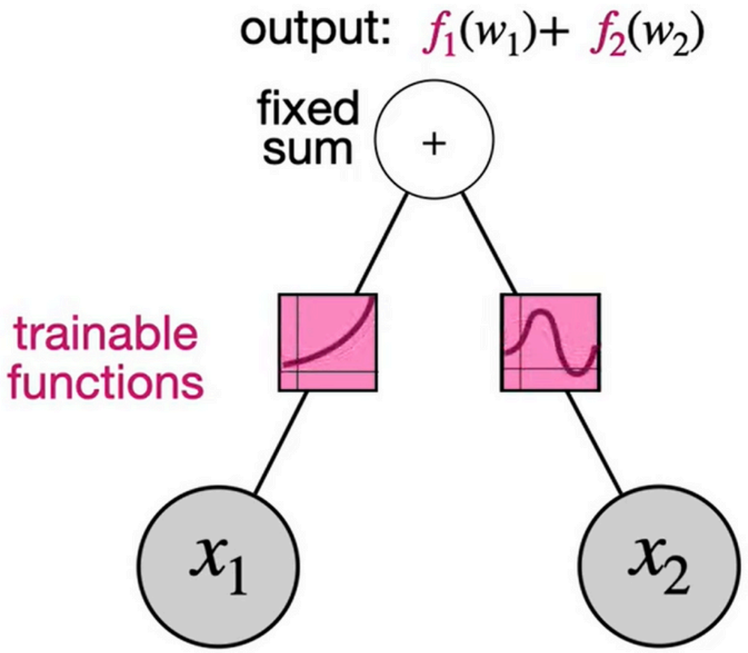
Training of Multi- Layer Perceptrons (MLPs)

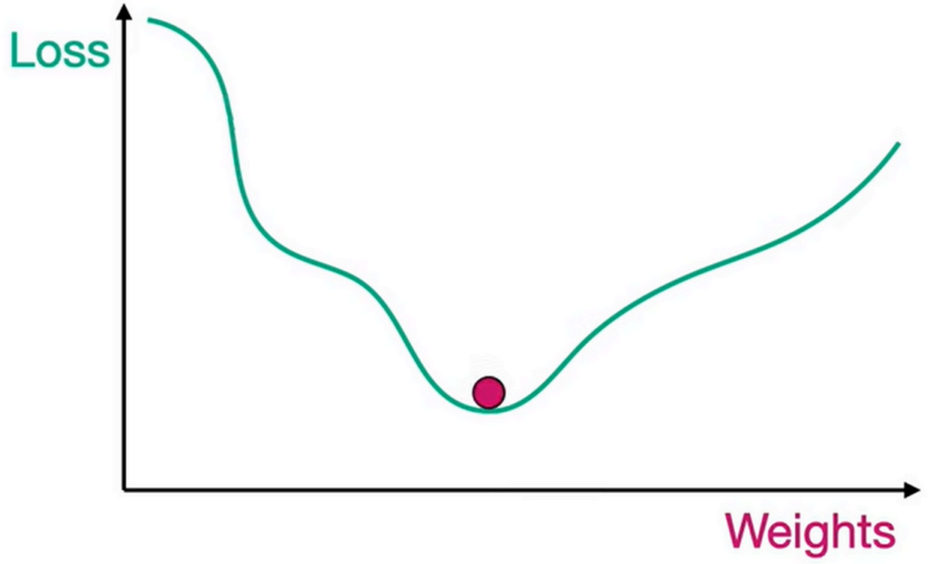




Change this weight using **gradient descent** and descent all the way to bottom to find the optimal low point

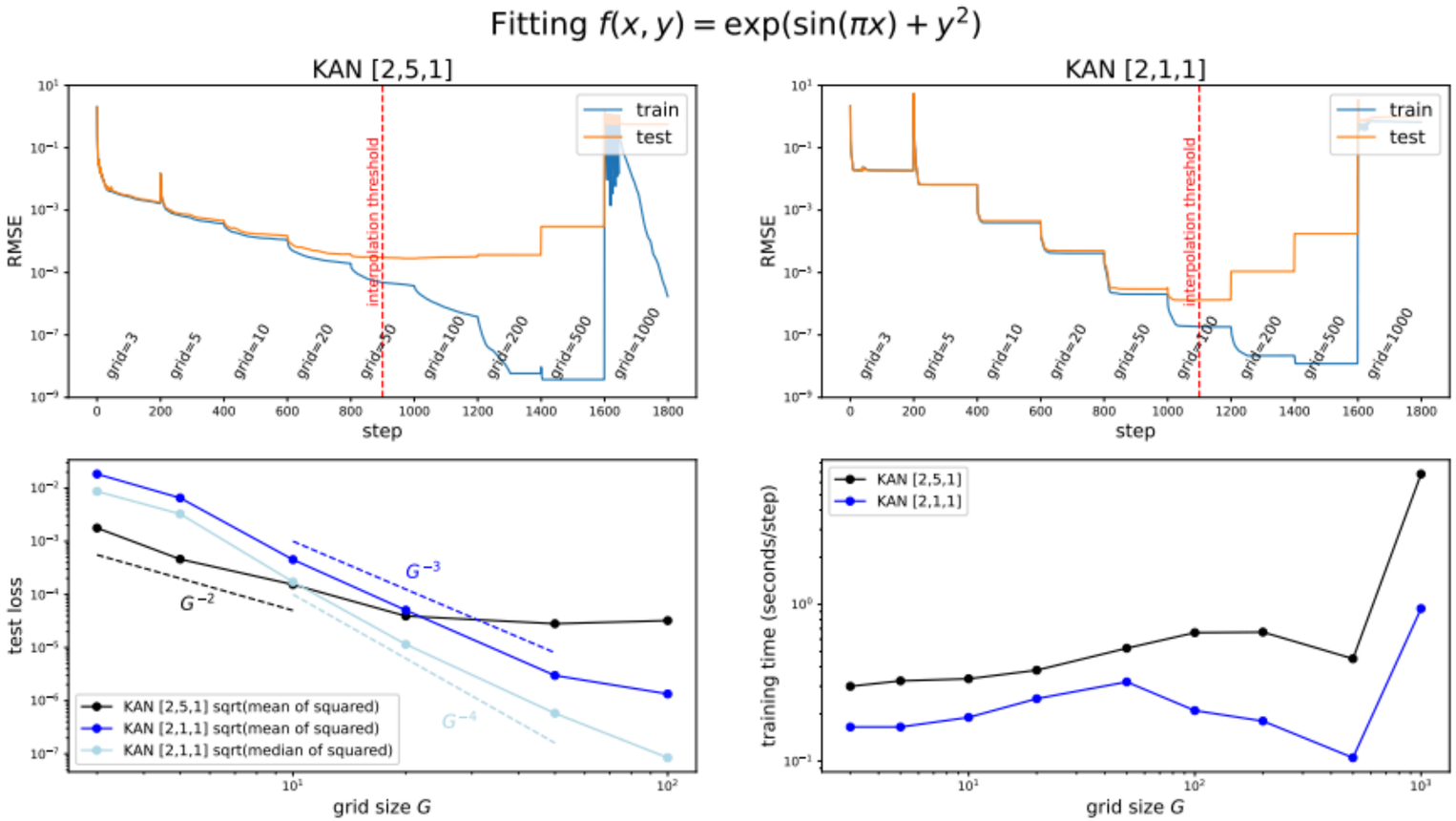
Training of Kolmogorov-Arnold Networks (KANs)





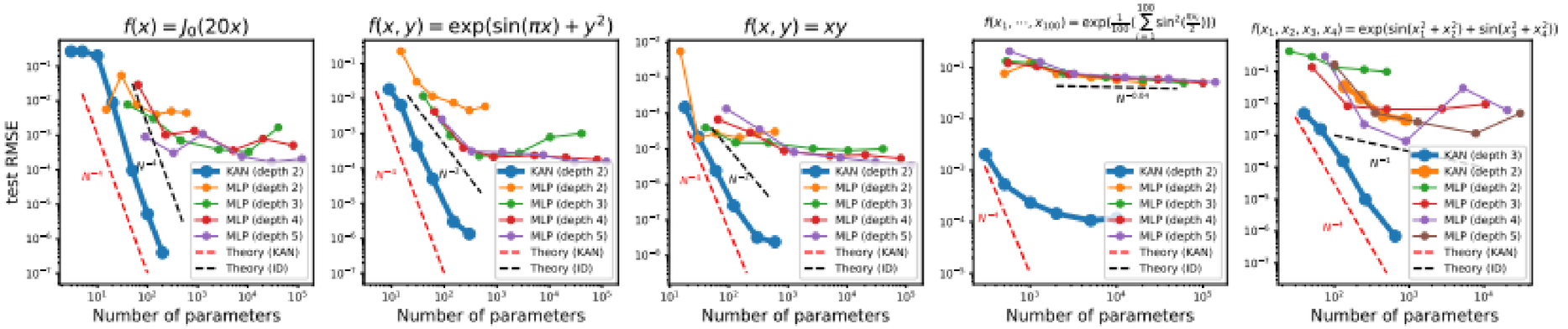
Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.

Kolmogorov-Arnold Networks (KANs) vs. Multi-Layer Perceptrons (MLPs)



We can make KANs more accurate by grid extension (fine-graining spline grids).

Accuracy of KAN: Advantage



Complexity in Practical Implementation: Disdvantage

Implementation of KAN on MNIST Dataset

```python
model = KAN(width=[784,10,1], grid=10, k=5, device=device)

def train_acc():
    return torch.mean((torch.round(model(dataset['train_input'])[:,0]) == dataset['train_label'][:,0]).type(dtype))

def test_acc():
    return torch.mean((torch.round(model(dataset['test_input'])[:,0]) == dataset['test_label'][:,0]).type(dtype))

results = model.fit(dataset, opt="LBFGS", steps=20, metrics=(train_acc, test_acc));
results['train_acc'][-1], results['test_acc'][-1]
```

```
checkpoint directory created: ./model
saving model version 0.0
| train_loss: 5.83e-02 | test_loss: 5.06e-02 | reg: 1.31e+02 | : 100%|| 20/20 [1:32:50<00:00, 278.5
saving model version 0.1

(0.9984998106956482, 0.9990543723106384)
```

(Model execution of KAN on MNIST dataset gives us 99% accuracy)

Time taken : 2 hours

Implementation of Artificial Neural Network on MNIST dataset

```python
model = keras.Sequential([
    keras.layers.Dense(100 , input_shape = (784 ,) , activation = "relu") ,
    keras.layers.Dense(2 , activation = "sigmoid")
])

model.compile(
    optimizer = "adam" ,
    loss = "sparse_categorical_crossentropy" ,
    metrics = ["accuracy"]
)

model.fit(X_train_norm , y_train_filtered , epochs= 10)
```

[27] ✓ 10.9s Python

```
Epoch 1/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9842 - loss: 0.0494
Epoch 2/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9986 - loss: 0.0039
Epoch 3/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9998 - loss: 0.0012
Epoch 4/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9992 - loss: 0.0011
Epoch 5/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9998 - loss: 4.0917e-04
Epoch 6/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9999 - loss: 2.2808e-04
Epoch 7/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 5.1786e-05
Epoch 8/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 1.0000 - loss: 6.0392e-05
Epoch 9/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 4.8837e-05
```

Accuracy of Model trained by ANN

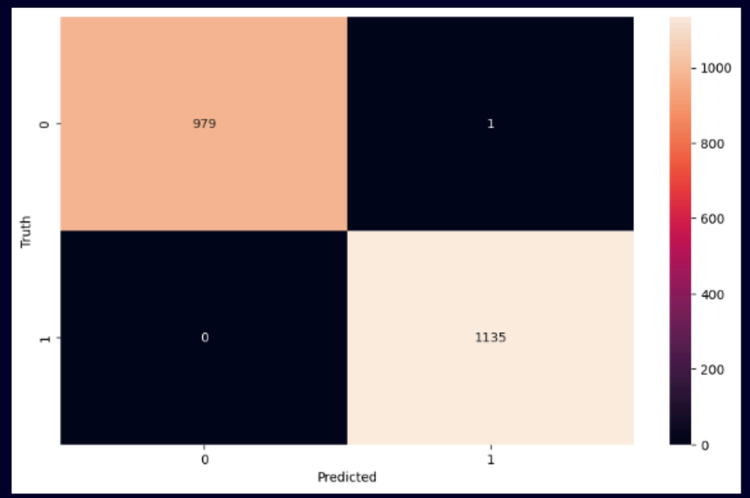
```
model.evaluate(X_test_norm , y_test_filtered)
```

```
67/67 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.0275e-04
[0.0016852435655891895, 0.9995272159576416]
```

```
y_predicted = model.predict(X_test_norm)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels = y_test_filtered , predictions= y_predicted_labels)
plt.figure(figsize=(10,6))
sns.heatmap(cm , annot = True , fmt = "d")
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

✓ 0.6s

```
67/67 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Text(95.72222222222221, 0.5, 'Truth')
```



Heart Disease Dataset

Data

This database contains 14 physical attributes based on physical testing of a patient. Blood samples are taken and the patient also conducts a brief exercise test. The "goal" field refers to the presence of heart disease in the patient. It is integer (0 for no presence, 1 for presence). In general, to confirm 100% if a patient has heart disease can be quite an invasive process, so if we can create a model that accurately predicts the likelihood of heart disease, we can help avoid expensive and invasive procedures.

Content

Attribute Information:

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
- target:0 for no presence of heart disease, 1 for presence of heart disease

Original Source: https://archive.ics.uci.edu/ml/datasets/Heart+Disease

Creators:

Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Implementation of KAN on the dataset

```python
model = KAN(width=[13 ,  1 ], grid = 3 , k=3, device=device)

def train_acc():
    return torch.mean((torch.round(model(dataset['train_input'])[:,0]) == dataset['train_label'][:,0]).type(dtype))

def test_acc():
    return torch.mean((torch.round(model(dataset['test_input'])[:,0]) == dataset['test_label'][:,0]).type(dtype))

results = model.fit(dataset, opt="LBFGS", steps=40, metrics=(train_acc, test_acc));
results['train_acc'][-1], results['test_acc'][-1]
```

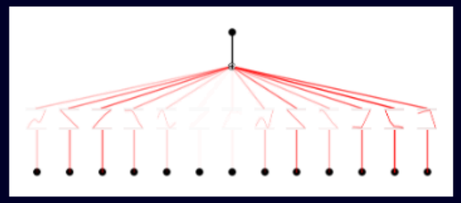
✓ 7.0s Python

```
checkpoint directory created: ./model
saving model version 0.0
| train_loss: 3.16e-01 | test_loss: 3.74e-01 | reg: 4.93e+00 | : 100%|| 40/40 [00:06<00:00,  5.95it
saving model version 0.1


(0.8866994976997375, 0.8399999737739563)
```

```python
model.plot()
```

✓ 2.2s Python



Automatic generated symbolic represenation

Automatic Symbolic Regression

```python
lib = ['x','x^2','x^3','x^4','exp','log','sqrt','tanh','sin','tan','abs']
model.auto_symbolic(lib=lib)
formula = model.symbolic_formula()[0][0]
ex_round(formula, 4)
```

✓ 3.3s Python

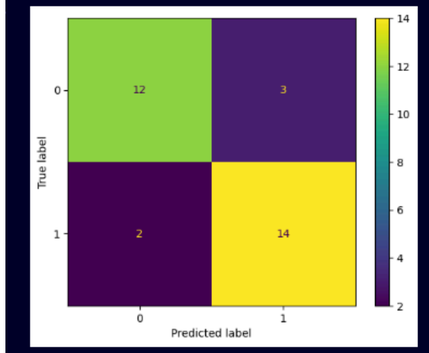
```
fixing (0,0,0) with x, r2=0.1705249696969986, c=1
fixing (0,1,0) with x, r2=1.0000004768371582, c=1
fixing (0,2,0) with sin, r2=0.9999949932098389, c=2
fixing (0,3,0) with x, r2=0.6921029090881348, c=1
fixing (0,4,0) with x, r2=0.7545654773712158, c=1
fixing (0,5,0) with exp, r2=1.0000008344650269, c=2
fixing (0,6,0) with x^2, r2=1.0000003576278687, c=2
fixing (0,7,0) with x, r2=0.10132782906293869, c=1
fixing (0,8,0) with x, r2=1.0000007152557373, c=1
fixing (0,9,0) with sin, r2=0.993678867816925, c=2
fixing (0,10,0) with x^2, r2=1.0000007152557373, c=2
fixing (0,11,0) with exp, r2=0.9979099035263062, c=2
fixing (0,12,0) with tan, r2=1.0000005960464478, c=3
saving model version 0.2
```

$$0.0015x_1 - 0.1631x_2 - 0.0022x_4 - 0.0004x_5 - 0.0004x_8 - 0.1665x_9 + 0.0984\left(0.5417 - x_{11}\right)^2 - 0.0113\left(1 - 0.7291x_7\right)^2 + 0.109\sin\left(0.5519x_{10} + 1.2585\right) + 0.2487\sin\left(5.8342x_3 + 9.7167\right) + 0.015\tan\left(7.199x_{13} - 7.3922\right) + 0.5981 - 0.0051e^{-7.392x_6} + 0.4626e^{-1.0056x_{12}}$$

Implementation of ANN on the dataset and its confusion matrix Display

```
# CODE HERE
ConfusionMatrixDisplay(confusion_matrix= cm).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x212016771f0>
```




```
# CODE HERE
print(classification_report(y_true = y_test,y_pred= y_predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.80      0.83        15
           1       0.82      0.88      0.85        16

    accuracy                           0.84        31
   macro avg       0.84      0.84      0.84        31
weighted avg       0.84      0.84      0.84        31
```

Author’s Note

I would like to thank everyone who's interested in KANs. When I designed KANs and wrote codes, I have math & physics examples (which are quite small scale!) in mind, so did not consider much optimization in efficiency or reusability. It's so honored to receive this unwarranted attention, which is way beyond my expectation. So I accept any criticism from people complaning about the efficiency and resuability of the codes, my apology. My only hope is that you find model.plot() fun to play with :).

For users who are interested in scientific discoveries and scientific computing (the orginal users intended for), I'm happy to hear your applications and collaborate. This repo will continue remaining mostly for this purpose, probably without signifiant updates for efficiency. In fact, there are already implmentations like efficientkan or fouierkan that look promising for improving efficiency.

For users who are machine learning focus, I have to be honest that KANs are likely not a simple plug-in that can be used out-of-the box (yet). Hyperparameters need tuning, and more tricks special to your applications should be introduced. For example, GraphKAN suggests that KANs should better be used in latent space (need embedding and unembedding linear layers after inputs and before outputs). KANRL suggests that some trainable parameters should better be fixed in reinforcement learning to increase training stability.

THANK YOU