Learning Day 27: Implementing RNN in Pytorch for time-series prediction

De Jun Huang · Follow
Published in dejunhuang
3 min read · May 13, 2021

Listen Share

A simple prediction task

- train model with 50 data points generated by sin function
- feed only 1 point and predict the next point, and feed the prediction for the next prediction, for approx. 50 times. x ->pred1->pred2 .... >pred50
- since there is only 1 point at each step, the feature_dim=1
- change data representation to bring batch forward as first dimension, therefore x = [batch, seq_len, feature_dim]. So in RNN setup, add new argument

  `batch_first=True`

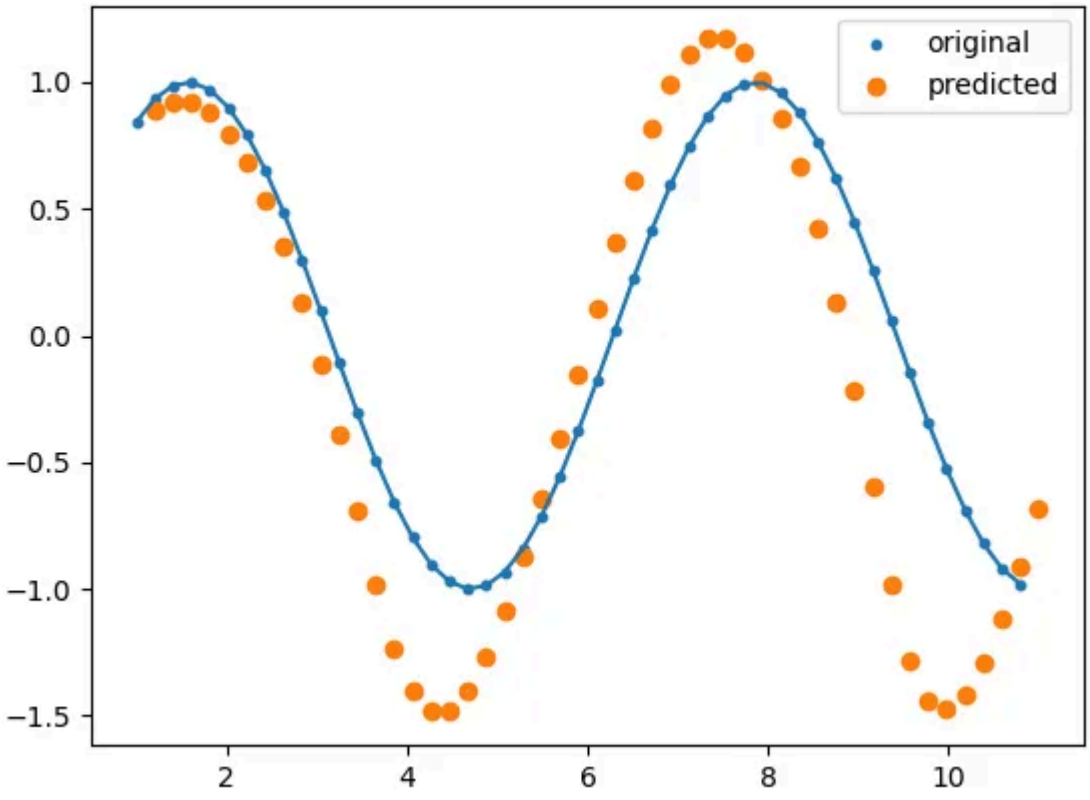
Implement RNN with nn.Module

- Largely similar to the way setting up CNN
- Involve a linear layer at the end to compress output from [batch, seq_len, mem_dim] to [seq_len, 1] so that output can compare with y
- hidden_size = memory_dim = 10

```
import torch
from torch import nn, optim
import numpy as np
import matplotlib.pyplot as plt

# number of points
num_time_steps = 50
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.rnn = nn.RNN(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=1,
            batch_first=True,
            # use batch_first for input with another data shape with
b first
        )

        # compress output to the same dim as y
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x, hidden_prev):
        out, hidden_prev = self.rnn(x, hidden_prev)
        # [1, seq, h] => [seq, h]  (batch=1)
        out = out.reshape(-1, hidden_size)  # stack batch and seq

        # linear layer so that output is not [seq,h] but [seq, 1]
        # so it is comparable with y, for loss calculation
        out = self.linear(out)  # [seq, h] => [seq, 1]
        out = out.unsqueeze(dim=0)  # => [1, seq, 1]
        return out, hidden_prev



model = Net()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr)

hidden_prev = torch.zeros(1, 1, hidden_size)  # [b, layer, mem_size]

for iter in range(6000):
    # randomly generate start point from 0 to 2
    start = np.random.randint(3, size=1)[0]
    # eg. from 0 to 10, create 50 points in between
    time_steps = np.linspace(start, start + 10, num_time_steps)
    data = np.sin(time_steps)
    data = data.reshape(num_time_steps, 1)

    # x: 49 points 0-49; y: 49 points 1-50
    x = torch.tensor(data[:-1]).float().reshape(1, num_time_steps -
1, 1)  # [b, seq_len, fea_len]
    y = torch.tensor(data[1:]).float().reshape(1, num_time_steps -
1, 1)  # [b, seq_len, fea_len]

    output, hidden_prev = model(x, hidden_prev)
    hidden_prev = hidden_prev.detach()
```

```
    # print(f"output {output.shape}, y {y.shape}")
    loss = criterion(output, y)
    model.zero_grad()
    # optimizer.zero_grad()
    # both zero_grad() are the same if model.parameters() is feed to
the same optimizer
    # only matters if multiple models using same optimizer or
multiple optims used for a model
    loss.backward()
    optimizer.step()

    if iter % 100 == 0:
        print(f"iteration: {iter}, loss {loss.item()}")

start = np.random.randint(3, size=1)[0]
time_steps = np.linspace(start, start+10, num_time_steps)
data = np.sin(time_steps)
data = data.reshape(num_time_steps, 1)
x = torch.tensor(data[:-1]).float().reshape(1, num_time_steps-1, 1)

preds = []
input_x = x[:, 0, :]  # select first point
for _ in range(x.shape[1]):
    input_x = input_x.reshape(1, 1, 1)  # reshape it for model
feeding
    pred, hidden_prev = model(input_x, hidden_prev)
    # print(pred.shape)
    # print(hidden_prev.shape)
    input_x = pred
    preds.append(pred.detach().numpy().ravel()[0])

x = x.data.numpy().ravel()
plt.scatter(time_steps[:-1], x.ravel(), s=10, label='original')
plt.plot(time_steps[:-1], x.ravel())

plt.scatter(time_steps[1:], preds, label='predicted')
plt.legend()
plt.show()
```



Results from above code implementation: Time series Prediction

Additional learning points

model.zero_grad() vs optimizer.zero_grad()

- the are the same if all model.parameters() were fed to the optimizer
- it only matters when multiple models are using the same optimizer, or multiple optimizers were used for different parts of a model (ref)

Prediction is made point by point

- RNN is flexible in the seq_len, ie. the number of data points/words
- How many points fed in = how many points being predicted
- It is interesting that the input seq_len can be as small as 1 as shown in the above example; do not have to feed in a bunch of data points for prediction

Question

- At the linear step, why batch size and seq_len can be stacked before linear layer?

```
(out = out.reshape(-1, hidden_size) # stack batch and seq
```

Ans: perhaps it doesn't matter.

- for b=1, [1, seq, h] => [seq, h], output = [seq, 1] after linear layer
- for b=3, [3, seq, h] => [3*seq, h], output = [3*seq, 1] after linear layer
- We just need to segregate each sequence again from the output to get the prediction for each sequence

Reference

link1

Daily Learning | Machine Learning | Data Science | Dejunhuang

Follow

Written by De Jun Huang

183 Followers · Editor for dejunhuang

More from De Jun Huang and dejunhuang

De Jun Huang in dejunhuang

Learning Day 57/Practical 5: Loss function—CrossEntropyLoss vs BCELoss in Pytorch; Softmax vs...

CrossEntropyLoss vs BCELoss

3 min read · Jun 12, 2021

141 3

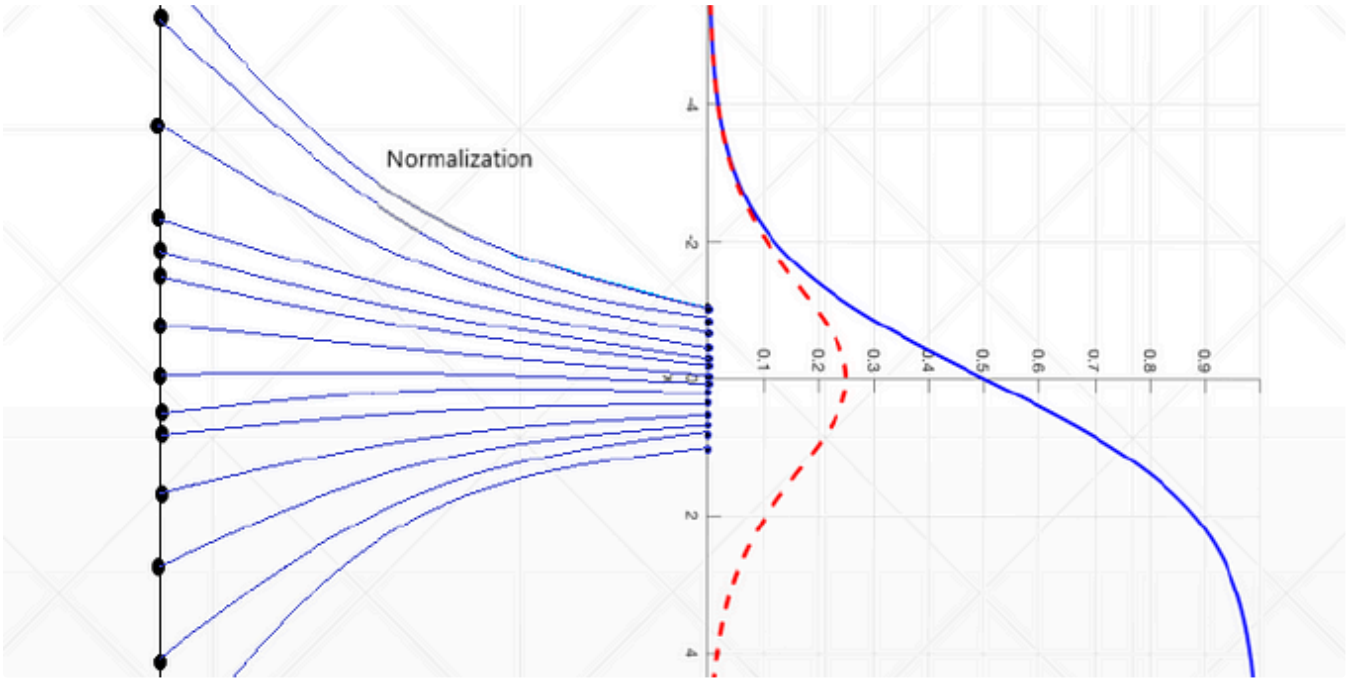
De Jun Huang in dejunhuang

Learning Day 22: What is nn.Module in Pytorch

Benefits of using nn.Module

2 min read · May 8, 2021

6



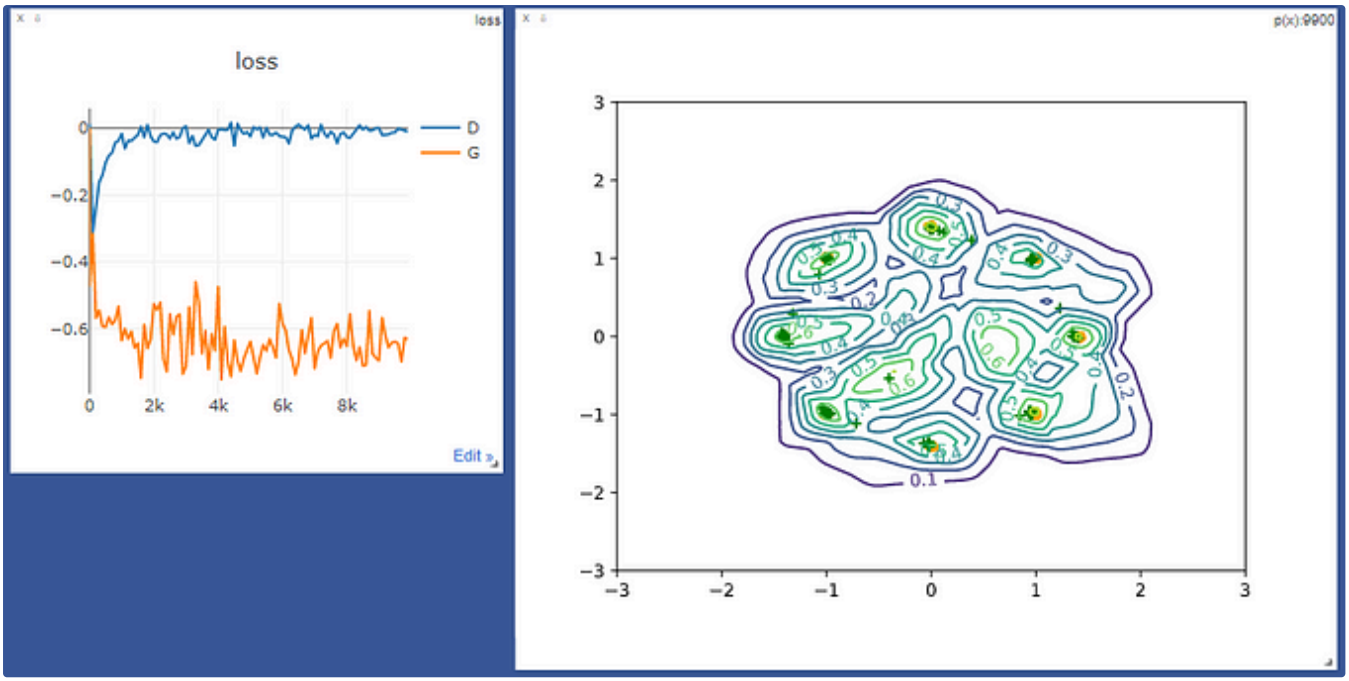
De Jun Huang in dejunhuang

Learning Day 20: Batch normalization concept and usage in Pytorch

Continue from Day 19, batch norm is another important element in a complete CNN architecture

2 min read · May 6, 2021

2



De Jun Huang in dejunhuang

Learning Day 41: Implementing GAN and WGAN in Pytorch

Implementing GAN

10 min read · May 27, 2021

6

See all from De Jun Huang

See all from dejunhuang

Recommended from Medium

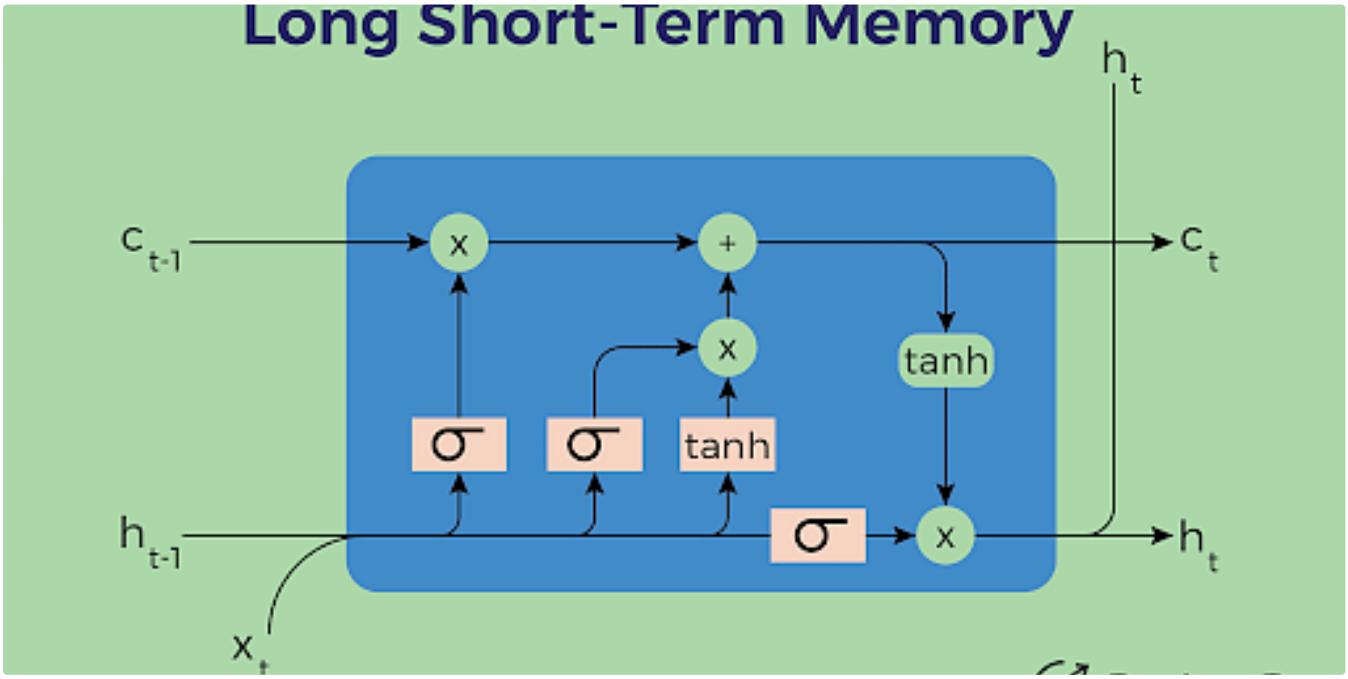
Abhimanyu HK

Understanding LSTM for Sequence Classification: A Practical Guide with PyTorch

Sequence classification is a common task in natural language processing, speech recognition, and bioinformatics, among other fields. Long...

2 min read · Mar 8, 2024

1



Palash Mishra

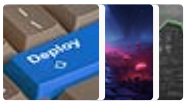
Understanding and Implementing LSTM Networks

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), have revolutionized the field of deep learning due to...

3 min read · Dec 29, 2023

2

Lists



Predictive Modeling w/ Python

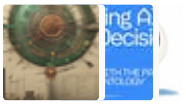
20 stories · 1229 saves

Practical Guides to Machine Learning

10 stories · 1484 saves

Natural Language Processing

1477 stories · 990 saves



data science and AI

40 stories · 169 saves

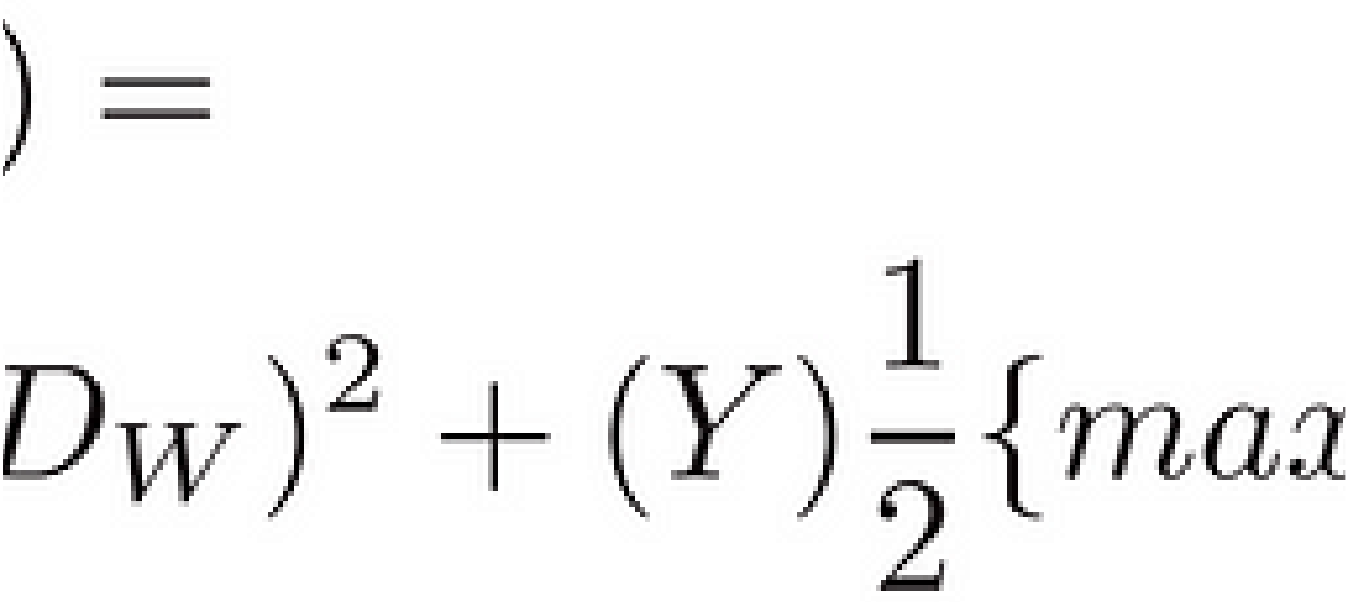
Michael Keith in Towards Data Science

Five Practical Applications of the LSTM Model for Time Series, with Code

How to implement an advanced neural network model in several different time series contexts

11 min read · Sep 22, 2023

207 1



Kelvin Teo Wei Min

Long Short Term Memory (LSTM) neural networks as an alternative to convolutional neural networks...

Convolutional neural networks (CNN) have achieved widespread use in computer vision processing, though the seminal publication of...

17 min read · Dec 26, 2023

3

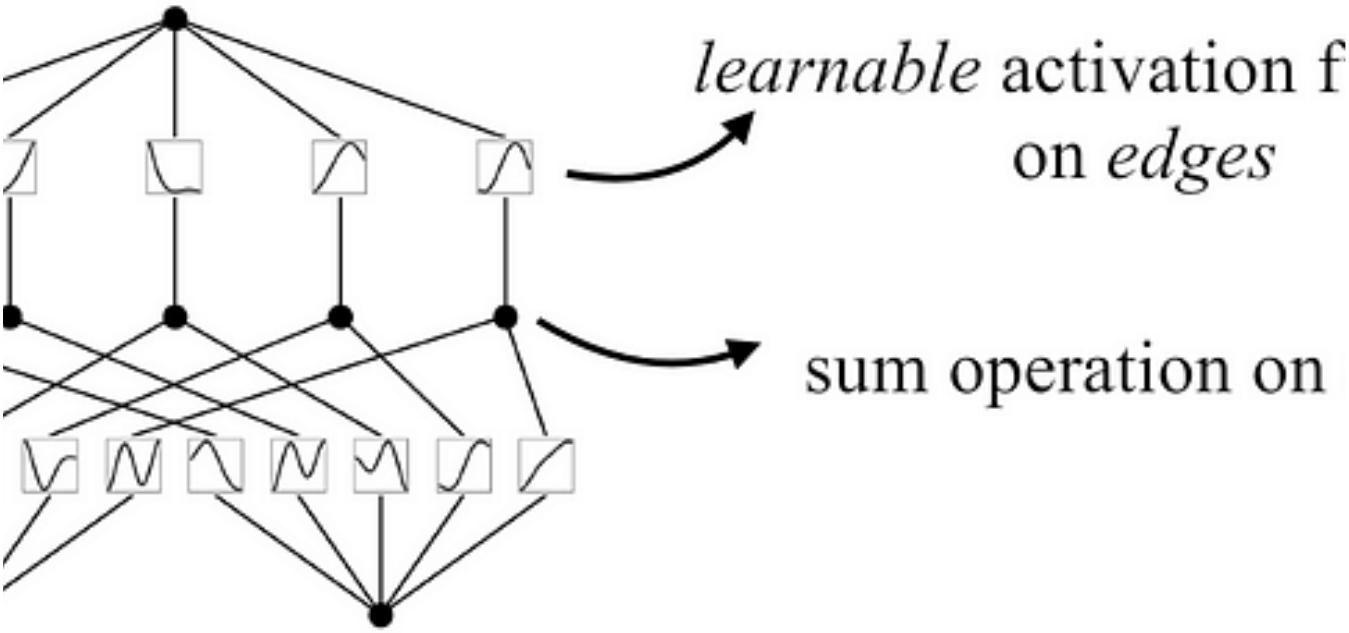
Kaan Aslan

Time Series Forecasting with a Basic Transformer Model in PyTorch

Time series forecasting is an essential topic that's both challenging and rewarding, with a wide variety of techniques available to...

10 min read · Jan 12, 2024

117 2



Siddharth Sudhakar in Accredian

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

This article will explore how KANs differ from traditional neural networks. We'll also examine the limitations that KANs must overcome.

6 min read · May 20, 2024

 1  

See more recommendations