LSTM

Statquest - machine learning - Long Short Term Memory

```
import torch ... import torch.nn as nn ... import torch.nn.functional as F ... from torch.optim import Adam
import lightning as L ... from torch.utils.data import TensorDataset, DataLoader

class LSTMbyHand (L.LightningModule):

    def __init__(self):

        super().__init__()

        mean = torch.tensor(0) ... std = torch.tensor(0.1)

        self.wlr1 = nn.Parameter(torch.normal(mean=mean, std=std), requires_grad=True
            wlr2  wpr2  wp2  wo2
            wpr1  wp1   wo1

        self.blr1 = nn.Parameter(torch.tensor(0), requires_grad=True)
            bpr1  bo1
            bp1

    def lstm_unit(self, input_value, long_memory, short_memory):

        long_remember_percent = torch.sigmoid((short_memory * self.wlr1)+(input_value * self.wlr2) + self.blr1)

        potential_remember_percent = torch.sigmoid((short_memory * self.wpr1)+(input_value * self.wpr2) + self.bpr1)

        potential_memory = torch.tanh((short_memory * self.wp1)+(input_value * self.wp2)+ self.bp1)

        updated_long_memory = ((long_memory * long_remember_percent)+(potential_remember_percent * potential_memory))

        output_percent = torch.sigmoid((short_memory * self.wo1)+(input_value * self.wo2)+ self.bo1)

        updated_short_memory = torch.tanh(updated_long_memory) * output_percent

        return([updated_long_memory, updated_short_memory])

    def forward(self, input):

        long_memory=0 ... short_memory=0

        day1 = input[0] ...

        long_memory, short_memory = self.lstm_unit(day1, long_memory, short_memory) ...

        return short_memory

    def configure_optimizers(self):

        return Adam(self.parameters())

    def training_step(self, batch, batch_idx):

        input_i, label_i = batch

        output_i = self.forward(input_i[0])

        loss = (output_i - label_i)**2
        self.log("train_loss", loss)

        if (label_i == 0): self.log("Out_0", output_i)

        else: self.log("Out_1", output_i)

        return loss
```

tensorboard

```
model = LSTMbyHand()
inputs = torch.tensor([[0, 0.5, 0.25, 1], [1, 0.5, 0.25, 1]]) ... labels = torch.tensor([0.1])
dataset = TensorDataset(inputs, labels) ... dataloader = DataLoader(dataset)
trainer = L.Trainer(max_epochs=2000)
trainer.fit(model, train_dataloaders=dataloader)
```

tensorboard

```
[
path_to_best_checkpoint = trainer.checkpoint_callback.best_model_path
trainer = L.Trainer(max_epochs=5000)
trainer.fit(model, train_dataloaders=dataloader, ckpt_path=path_to_best_checkpoint)


class LightningLSTM(L.LightningModule):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=1)
    def forward(self, input):
        input_trans = input.view(len(input), 1)
        lstm_out, temp = self.lstm(input_trans)
        prediction = lstm_out[-1]
        return prediction
    def configure_optimizers(self):
        ''
    def training_step(self, batch, batch_idx):
        ''

model = LightningLSTM()
trainer = L.Trainer(max_epochs=300, log_every_n_steps=2)
trainer.fit(model, train_dataloaders=dataloader)
```