CoreAlgorithm

March 2, 2023

1 Core Machine Learning Algorithm Realization by PyTorch

Every strings under '' (like '<Machine Tools Input factor Data>') must be changed to your own value before running!

1.1 make sure you have PyTorch installed before running this script

1.2 import training data and torch package

[ ]:
```
import torch
# the format of machine_input_data and tool_value are tensor_format!
machine_input_data = torch.Tensor('<Machine Tools Input factor Data>')
tool_value = torch.Tensor('<Machine Tool predicting value>')
```

1.3 Define Neural Network Training Model

We use linear regression model which can be expressed by

$$\hat{Predict} = weight * machine\_input\_data + bias$$

[ ]:
```
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        # in_features stands for neural network nodes input number
        # output_features stands for neural network predicting nodes output
        in_features = 65
        out_features = 1
        self.linear = torch.nn.Linear(in_features, out_features, bias=True)

    def forward(self, x):
        tool_value_pred = self.linear(x)
        return tool_value_pred


model = LinearModel()
```

1.4 Define Neural Network loss function and Optimization Algorithm

Using MSE (mean quared error) to measure the loss and its math expression as below:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2$$

Implements stochastic gradient descent as gradient descent algorithm (optionally with momentum)

[ ]:
```
# MSE loss function
criterion = torch.nn.MSELoss(reduction='sum')
# lr = learning_rate
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

[ ]:
```
for epoch in range('<Define Iteration Epoch Times>'):
    tool_value_pred = model(machine_input_data)
    loss = criterion(tool_value_pred, tool_value)
    print(epoch, loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

1.5 Output Model Training Results and Test New Data

[ ]:
```
# Output weight and bias
print('w = ', model.linear.weight)
print('b = ', model.linear.bias)

# Test Model
machine_input_data = torch.Tensor('<relative machine signal and other␣
 ↪constant>')
tool_value_pred = model(machine_input_data)
print('tool_value_pred = ', tool_value_pred.data)
```