In [1]:
```
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,accuracy_score,classification_report
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import pandas
```

In [2]:
```
# CHOOSE DATASET

# Binary classification dataset
diabetes = datasets.load_diabetes(as_frame=True)

# Regression dataset
#data = datasets.fetch_openml(name="boston",version=1, as_frame=True)

X = diabetes.data.values
y = diabetes.target.values

print("Shape:", X.shape)

print(diabetes.data.head(), "\n \n")# first rows of features
print(diabetes.target.head()) # first rows of target
```

```
Shape: (442, 10)
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6
0 -0.002592  0.019907 -0.017646
1 -0.039493 -0.068332 -0.092204
2 -0.002592  0.002861 -0.025930
3  0.034309  0.022688 -0.009362
4 -0.002592 -0.031988 -0.046641


0    151.0
1     75.0
2    141.0
3    206.0
4    135.0
Name: target, dtype: float64
```

In [3]:
```
#train test spliting
test_size=0.2
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=test_size, random_state=42)
```

In [4]:
```
# Standardize features
scaler=StandardScaler()
Xtr= scaler.fit_transform(Xtr)
Xte= scaler.transform(Xte)
```

A fixed seed was added to this code to ensure the reproducibility of the analysis. This allowed for manual tuning of the hyperparameters to achieve better model training.

In [5]:
```
import random

seed = 42
torch.manual_seed(seed)
```

```
np.random.seed(seed)
random.seed(seed)

# Para GPU
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)

# Tornar CUDA determinístico
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

In [6]:
```
class MLP(nn.Module):
    def __init__(self, input_size, output_size=1, dropout_prob=0.5):
        super(MLP, self).__init__()

        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 64)
        self.out = nn.Linear(64, output_size)

        self.dropout = nn.Dropout(p=dropout_prob)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)

        x = F.relu(self.fc2(x))
        x = self.dropout(x)

        x = F.relu(self.fc3(x))
        x = self.dropout(x)

        x = F.relu(self.fc4(x))
        x = self.dropout(x)

        x = self.out(x)
        return x
```

In [7]:
```
num_epochs=190
lr=0.01
dropout=0.25
batch_size=128
```

This model was trained on the GPU and then transferred to the CPU for use with NumPy. Given the low number of parameters (i.e., the model's low complexity), the time required on the CPU was similar to that on the GPU.

In [8]:
```
# Model, Loss, Optimizer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#device = "cpu"  # force to use CPU
print(device)
```

```
cuda
```

In [9]:
```
Xtr = torch.tensor(Xtr, dtype=torch.float32).to(device)
ytr = torch.tensor(ytr, dtype=torch.float32).to(device)
Xte = torch.tensor(Xte, dtype=torch.float32).to(device)
yte = torch.tensor(yte, dtype=torch.float32).to("cpu")

# Wrap Xtr and ytr into a dataset
train_dataset = TensorDataset(Xtr, ytr)

# Create DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

In [10]:
```
model = MLP(input_size=Xtr.shape[1], dropout_prob=dropout).to(device)
criterion = nn.BCEWithLogitsLoss()  # for binary classification
criterion = nn.MSELoss() #for regression
optimizer = optim.Adam(model.parameters(), lr=lr)
```

The model was implemented as a fully connected neural network (MLP) with four hidden layers of 64 neurons each, using ReLU activation functions. The network takes the input features of the dataset and outputs a single value for regression (Diabetes Progression).

In [11]:
```
# Training Loop
import time
start_time = time.time()
for epoch in range(num_epochs):
```

```
    model.train()
    epoch_loss = 0.0

    for batch_x, batch_y in train_dataloader:
        batch_x = batch_x.to(device)
        batch_y = batch_y.to(device)

        logits = model(batch_x)
        loss = criterion(logits, batch_y.view(-1, 1))

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

        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(train_dataloader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")

end_time = time.time()
print(f"Training time: {end_time - start_time:.2f} seconds")
```

```
Epoch [1/190], Loss: 29648.7161
Epoch [2/190], Loss: 27738.2480
Epoch [3/190], Loss: 19526.0052
Epoch [4/190], Loss: 10916.4779
Epoch [5/190], Loss: 7908.8825
Epoch [6/190], Loss: 6315.4227
Epoch [7/190], Loss: 6881.3011
Epoch [8/190], Loss: 5394.8262
Epoch [9/190], Loss: 5833.4985
Epoch [10/190], Loss: 4486.8579
Epoch [11/190], Loss: 4801.0705
Epoch [12/190], Loss: 4380.2632
Epoch [13/190], Loss: 4705.3464
Epoch [14/190], Loss: 4574.8371
Epoch [15/190], Loss: 4115.9676
Epoch [16/190], Loss: 4263.3736
Epoch [17/190], Loss: 4220.0199
Epoch [18/190], Loss: 3679.3091
Epoch [19/190], Loss: 3745.2568
Epoch [20/190], Loss: 3896.5340
Epoch [21/190], Loss: 4014.5900
Epoch [22/190], Loss: 3893.0127
Epoch [23/190], Loss: 3573.5227
Epoch [24/190], Loss: 3891.4432
Epoch [25/190], Loss: 3775.5807
Epoch [26/190], Loss: 3753.8115
Epoch [27/190], Loss: 3697.5588
Epoch [28/190], Loss: 3638.7832
Epoch [29/190], Loss: 4002.4415
Epoch [30/190], Loss: 3921.0071
Epoch [31/190], Loss: 3542.5956
Epoch [32/190], Loss: 3535.1024
Epoch [33/190], Loss: 3598.4771
Epoch [34/190], Loss: 3983.6483
Epoch [35/190], Loss: 3522.1357
Epoch [36/190], Loss: 3819.0868
Epoch [37/190], Loss: 3664.6637
Epoch [38/190], Loss: 3815.6850
Epoch [39/190], Loss: 3901.3689
Epoch [40/190], Loss: 3601.0779
Epoch [41/190], Loss: 3759.6592
```

```
Epoch [42/190], Loss: 3391.2233
Epoch [43/190], Loss: 3242.0986
Epoch [44/190], Loss: 3577.6099
Epoch [45/190], Loss: 3480.0977
Epoch [46/190], Loss: 3555.8261
Epoch [47/190], Loss: 3476.3621
Epoch [48/190], Loss: 3268.5854
Epoch [49/190], Loss: 3618.4499
Epoch [50/190], Loss: 3584.3310
Epoch [51/190], Loss: 3675.5367
Epoch [52/190], Loss: 3722.2685
Epoch [53/190], Loss: 3227.3150
Epoch [54/190], Loss: 2974.1118
Epoch [55/190], Loss: 3562.6582
Epoch [56/190], Loss: 3198.2245
Epoch [57/190], Loss: 3238.8750
Epoch [58/190], Loss: 3376.6782
Epoch [59/190], Loss: 3346.6515
Epoch [60/190], Loss: 3468.1538
Epoch [61/190], Loss: 3152.5792
Epoch [62/190], Loss: 3484.5428
Epoch [63/190], Loss: 3397.3236
Epoch [64/190], Loss: 3454.3319
Epoch [65/190], Loss: 3955.6242
Epoch [66/190], Loss: 3623.3734
Epoch [67/190], Loss: 3310.0218
Epoch [68/190], Loss: 3456.5499
Epoch [69/190], Loss: 3696.4705
Epoch [70/190], Loss: 3757.5662
Epoch [71/190], Loss: 3274.0444
Epoch [72/190], Loss: 3489.4048
Epoch [73/190], Loss: 3018.0824
Epoch [74/190], Loss: 3315.5653
Epoch [75/190], Loss: 3203.5889
Epoch [76/190], Loss: 3338.0278
Epoch [77/190], Loss: 3350.0305
Epoch [78/190], Loss: 3378.0791
Epoch [79/190], Loss: 3423.2143
Epoch [80/190], Loss: 3527.6632
Epoch [81/190], Loss: 3268.2509
Epoch [82/190], Loss: 3227.4768
```

```
Epoch [83/190], Loss: 3589.3934
Epoch [84/190], Loss: 3299.7594
Epoch [85/190], Loss: 3385.5459
Epoch [86/190], Loss: 3498.3327
Epoch [87/190], Loss: 3477.1771
Epoch [88/190], Loss: 3174.3293
Epoch [89/190], Loss: 3714.6216
Epoch [90/190], Loss: 3343.3033
Epoch [91/190], Loss: 3414.0710
Epoch [92/190], Loss: 3248.5864
Epoch [93/190], Loss: 3488.2712
Epoch [94/190], Loss: 3373.2797
Epoch [95/190], Loss: 3372.9124
Epoch [96/190], Loss: 3554.2501
Epoch [97/190], Loss: 3368.5915
Epoch [98/190], Loss: 3215.7687
Epoch [99/190], Loss: 3260.8147
Epoch [100/190], Loss: 3192.8174
Epoch [101/190], Loss: 3423.2056
Epoch [102/190], Loss: 2991.1534
Epoch [103/190], Loss: 3152.7769
Epoch [104/190], Loss: 3303.9213
Epoch [105/190], Loss: 3282.2486
Epoch [106/190], Loss: 3331.6805
Epoch [107/190], Loss: 3279.8656
Epoch [108/190], Loss: 3262.2849
Epoch [109/190], Loss: 3125.2458
Epoch [110/190], Loss: 3407.5113
Epoch [111/190], Loss: 3498.1632
Epoch [112/190], Loss: 3553.5989
Epoch [113/190], Loss: 3089.8211
Epoch [114/190], Loss: 3075.9001
Epoch [115/190], Loss: 3539.4534
Epoch [116/190], Loss: 2961.9173
Epoch [117/190], Loss: 3655.1819
Epoch [118/190], Loss: 3049.0596
Epoch [119/190], Loss: 3615.5492
Epoch [120/190], Loss: 3256.5053
Epoch [121/190], Loss: 3308.7235
Epoch [122/190], Loss: 3240.1512
Epoch [123/190], Loss: 3193.3521
```

```
Epoch [124/190], Loss: 3216.8752
Epoch [125/190], Loss: 3073.2205
Epoch [126/190], Loss: 3247.6636
Epoch [127/190], Loss: 3242.5971
Epoch [128/190], Loss: 3164.6580
Epoch [129/190], Loss: 3109.4061
Epoch [130/190], Loss: 3200.9823
Epoch [131/190], Loss: 2903.9763
Epoch [132/190], Loss: 3002.0817
Epoch [133/190], Loss: 3106.7635
Epoch [134/190], Loss: 3022.0481
Epoch [135/190], Loss: 2955.9606
Epoch [136/190], Loss: 2726.1379
Epoch [137/190], Loss: 2718.2030
Epoch [138/190], Loss: 2962.2109
Epoch [139/190], Loss: 2859.1110
Epoch [140/190], Loss: 2965.7663
Epoch [141/190], Loss: 2804.8240
Epoch [142/190], Loss: 3106.8247
Epoch [143/190], Loss: 2979.2811
Epoch [144/190], Loss: 3120.4154
Epoch [145/190], Loss: 3010.3355
Epoch [146/190], Loss: 2924.7756
Epoch [147/190], Loss: 3123.8313
Epoch [148/190], Loss: 3109.5750
Epoch [149/190], Loss: 3098.7642
Epoch [150/190], Loss: 2857.3700
Epoch [151/190], Loss: 2996.3384
Epoch [152/190], Loss: 3129.6170
Epoch [153/190], Loss: 2759.2476
Epoch [154/190], Loss: 3000.5859
Epoch [155/190], Loss: 2925.3764
Epoch [156/190], Loss: 2861.1976
Epoch [157/190], Loss: 2986.6821
Epoch [158/190], Loss: 2743.7799
Epoch [159/190], Loss: 3126.5764
Epoch [160/190], Loss: 2914.5562
Epoch [161/190], Loss: 2880.0532
Epoch [162/190], Loss: 2857.0591
Epoch [163/190], Loss: 2817.7529
Epoch [164/190], Loss: 2701.1991
```

```
Epoch [165/190], Loss: 3057.9538
Epoch [166/190], Loss: 2827.3732
Epoch [167/190], Loss: 2802.8832
Epoch [168/190], Loss: 3131.7845
Epoch [169/190], Loss: 3075.6080
Epoch [170/190], Loss: 2931.3674
Epoch [171/190], Loss: 2838.9221
Epoch [172/190], Loss: 3041.7693
Epoch [173/190], Loss: 3094.6465
Epoch [174/190], Loss: 3209.8215
Epoch [175/190], Loss: 3203.1895
Epoch [176/190], Loss: 3175.8590
Epoch [177/190], Loss: 2773.8187
Epoch [178/190], Loss: 3101.5061
Epoch [179/190], Loss: 3159.9993
Epoch [180/190], Loss: 2911.1960
Epoch [181/190], Loss: 2810.7509
Epoch [182/190], Loss: 2745.5351
Epoch [183/190], Loss: 2873.0893
Epoch [184/190], Loss: 3086.1829
Epoch [185/190], Loss: 3048.7624
Epoch [186/190], Loss: 2893.8034
Epoch [187/190], Loss: 2915.0261
Epoch [188/190], Loss: 2572.4602
Epoch [189/190], Loss: 2886.5369
Epoch [190/190], Loss: 2821.7037
Training time: 1.99 seconds
```

In [12]:

```
y_pred=model(Xte).cpu()  # só nesta altura volta ao CPU
#print(f'ACC:{accuracy_score(yte.detach().numpy(),y_pred.detach().numpy()>0.5)}') #classification

print(f'MSE:{mean_squared_error(yte.detach().numpy(),y_pred.detach().numpy())}') #regression
```

```
MSE:2814.833251953125
```

After tuning the hyperparameters, the best configuration was: num_epochs = 200, lr = 0.02, dropout = 0.2, and batch_size = 128.A dropout with a probability of 0.2 was applied after each layer to mitigate overfitting.It order to validate the model the MSE was computed. The model was evaluated using the Mean Squared Error (MSE), which resulted in a high value of 2815. This indicates that, despite the training and parameter optimization, the current approach is insufficient to accurately predict Diabetes Progression. The high error may be attributed to the limited size of the dataset, the low number of features. In comparison with previous approaches, such as ANFIS or TSK models, this approach exhibited the worst performance.

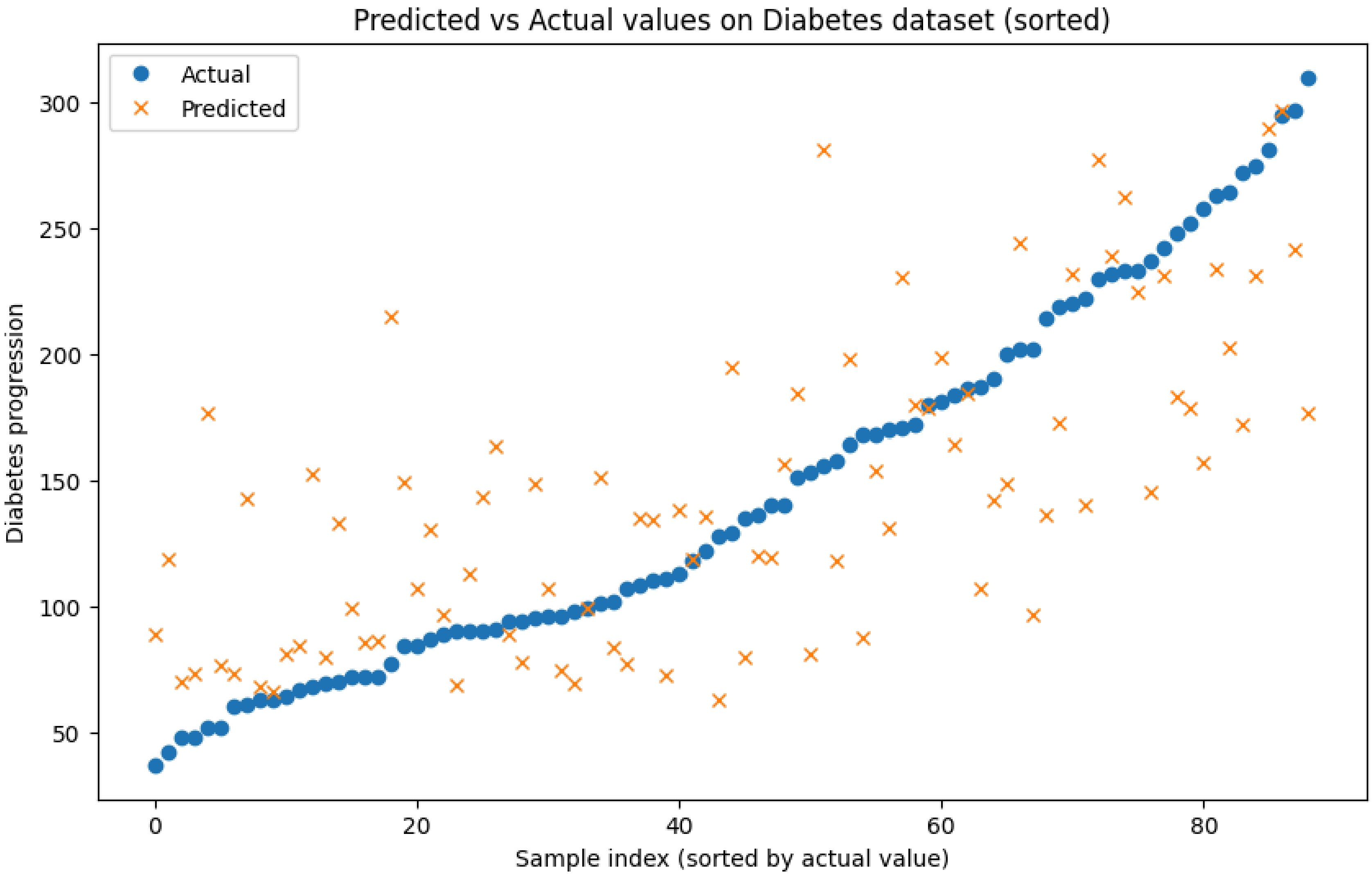
In [13]:
```
# Plot predictions vs actual
# Converter y_pred para numpy e flatten
y_pred_np = y_pred.detach().numpy().flatten()
yte_np = yte.detach().numpy()

# Obter índices que ordenam yte
sort_idx = np.argsort(yte_np)

# Ordenar yte e y_pred segundo esses índices
yte_sorted = yte_np[sort_idx]
y_pred_sorted = y_pred_np[sort_idx]

# Plot
plt.figure(figsize=(10,6))
plt.plot(range(len(yte_sorted)), yte_sorted, label="Actual", marker="o", linestyle='')
plt.plot(range(len(y_pred_sorted)), y_pred_sorted, label="Predicted", marker="x", linestyle='')

plt.xlabel("Sample index (sorted by actual value)")
plt.ylabel("Diabetes progression")
plt.title("Predicted vs Actual values on Diabetes dataset (sorted)")
plt.legend()
plt.show()
```


Predicted vs Actual values on Diabetes dataset (sorted)
Actual
Predicted
300
250
200
150
100
50
Diabetes progression
0
20
40
60
80
Sample index (sorted by actual value)