rASDS 5305 Final Project # 4: Optimizer COmparison on Deep Network

04/11/2025

Group Name: sp25_BerKyd

Group Members:

- Henry Berrios #1001392315
- LeMaur Kydd #1001767382

Optimizer Comparisons Google Colab

Overview:

For round 4, the deep neural network from Round 3 was reused, and compared the effects of different optimization strategies on model performance and training time. Specifically, Gradient Descent, Stochastic Gradient Descent and SGD with Momentum were used. Performance on the SMILES dataset was evaluated using accuracy, precision, recall, F1-score and test loss.

B. Gradient Descent (GD):

```
# using the full dataset as a single batch (gradinet descent)
gd_loader = DataLoader(train_dataset, batch_size = len(train_dataset),
shuffle = True)
```

```
# intializing the model
gd_model = DeepNetwork(input_size = X_train.shape[1] * X_train.shape[2],
                       hidden_layers = best_config[0],
                       hidden_units = best_config[0],
                       output_size = y_train.shape[1]).to(device)
```

```
# loss function cross entropy for multi-class classification
criterion = nn.CrossEntropyLoss()
```

```
# optimizer using vanilla gradinet descent (single full batch)
optimizer = optim.SGD(gd_model.parameters(), lr = 0.001)
```

Model performance:

- Batch size: Full dataset (5, 548)
- Training time: ~96.84 seconds
- Test Accuracy: 0.3453
- Precision: 0.1193
- Recall: 0.3453
- F1 Score: 0.1773
- Test Loss on the final: 7.11
- Test Loss on the final: 6.06

Plot Batch size vs. Test accuracy:

Overall summary:

- Using the entire training set was stable and predictable during training, but it was computationally expensive and generalized poorly
- Full-batch GD has the advantage of precise gradient calculation, but this also makes it more prone to overfitting and less responsive to noisy updates. This can be seen especially with our dataset where it can generally be imbalance or complex.

C. Stochastic Gradient Descent (SGD)

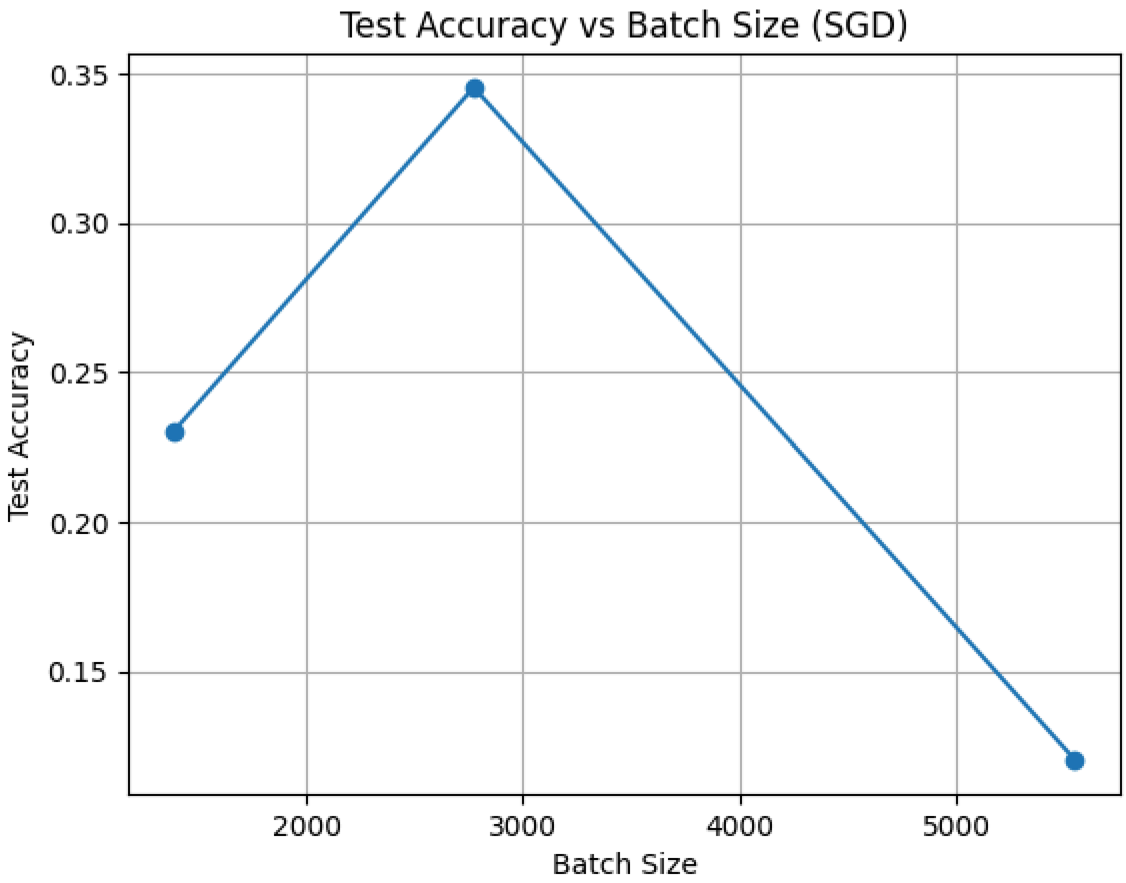
```
# creating a list of batch sizes (full batch, 1/2, 1/4 and so on)
batch_size = [len(train_dataset) // (2 ** i) for i in range(5)]
sgd_results = []
```

```
# looping over each batch size
for bs in batch_size:
  print(f"Training model with batch size {bs}")

  # creating a DataLoader with the given batch size
  sgd_loader = DataLoader(train_dataset, batch_size = bs, shuffle = True)

  # intializing the model for each individual run
  sgd_model = DeepNetwork(input_size = X_train.shape[1] *
X_train.shape[2],
                          hidden_layers = best_config[0],
                          hidden_units = best_config[1],
                          output_size = y_train.shape[1]).to(device)
```

Accuracy Trends with Batch Size:

Batch Size	Accuracy	Final Test Loss	Training Time (s)
5548	0.1204	29.23	317.39
2774	0.3453	45.42	321.18
1387	0.2300	1.70	327.68

Plot Batch size vs. Test accuracy:



Overall Summary:

- We tested the batch sizes from 5548 down to 1387, halving at each step. The smaller the batch size, the longer the training time, but generalization and accuracy seemed to improve. Our best accuracy was ~34.53% at batch size 2774, which matches what we got for GD, but at much lower final test loss.

- Mini-batch SGD adds noise to the optimization path, which helps the model escape local minima and converge more efficiently. However, the smaller batch sizes also increase training time. Best performance seemed to be around batch size 2774, which maintained GD-level accuracy with better test loss.

C. Stochastic Gradient Descent with Momentum (SGD w/ Momentum)

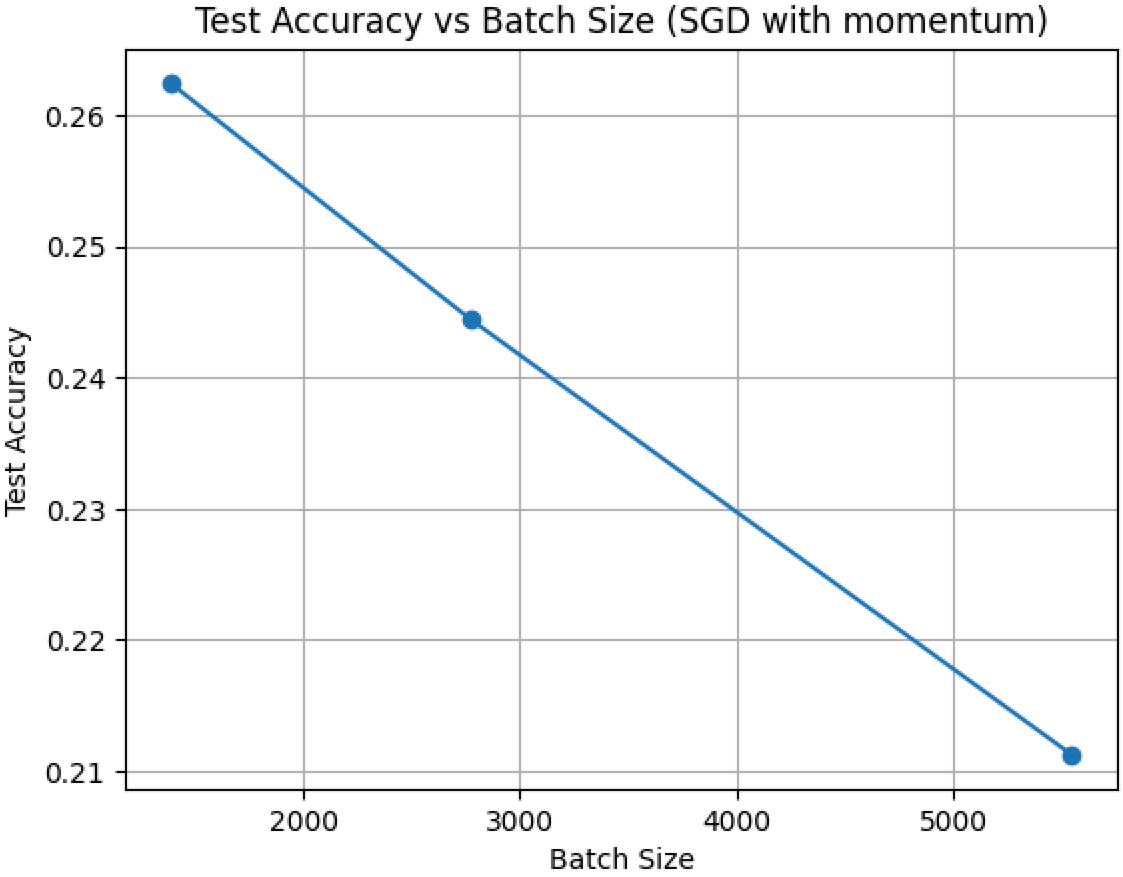
```
# various portions of this code block were filled in with autofill AI

batch_sizes = [len(train_dataset) // (2 ** i) for i in range(3)]
momentum_results = []
```

```
# various portions of this code block were filled in with autofill AI

# looping over batch sizes
for batch_size in batch_sizes:
 gd_loader = DataLoader(train_dataset, batch_size = batch_size, shuffle = True)

 gd_model = DeepNetwork(
     input_size = X_train.shape[1] * X_train.shape[2],
     hidden_layers = best_config[0],
     hidden_units = best_config[1],
     output_size = y_train.shape[1]
 ).to(device)
```

Accuracy Trends with Batch Size:

Batch Size	Accuracy	Final Test Loss	Training Time (s)
5548	0.2112	157.75	317.17
2774	0.2444	74.05	321.49
1387	0.2624	1.91	327.64

Plot Batch size vs. Test accuracy:



Overall Summary:

- We tested the batch sizes from 5548 down to 1387, halving at each step. The training time did not seem to change significantly as the batch size got smaller, but model accuracy slowly got better as batch size shrunk. Our best accuracy was ~26.24% at batch size 1387.