ASDS 5305 Final Project Datasets

3/7/2025

Group Name: sp25_BerKyd

Group Members:

- Henry Berrios #1001392315
- LeMaur Kydd #1001767382

Shallow Network Google Colab

Deep Network Google Colab

Dataset 1

Title: Drug SMILES Strings and Classifications

Source: Research Paper Supporting Information SMILES LINK

Shallow Network Analysis

A. Create a Shallow Neural Network

A shallow neural network with the following parameters was used:

- **Hidden layers: 1 layer**
- **Neurons per layer: 2, 8, 32, and 128**
- **Activation Function: ReLU**
- **Loss Function: CrossEntropyLoss**
- **Optimizer: Adam** (learning rate = **0.001**)
- **Batch Size: 256** (optimized for T4 GPU)
- **Training and Testing Dataset Shape:**
  - **X_train**: (5548, 399, 128)
  - **y_train**: (5548, 5)
  - **X_test**: (1387, 399, 128)
  - **y_test**: (1387, 5)

```
# various snippets of code in this block were autofilled in using AI

class ShallowNN(nn.Module):
 def __init__(self, input_dim, hidden_dim, output_dim, embedding_dim =
```

```
128):
    super(ShallowNN, self).__init__()
    self.fc1 = nn.Linear(input_dim * embedding_dim, hidden_dim)
    self.fc2 = nn.Linear(hidden_dim, output_dim)

 def forward(self, x):
    x = x.view(x.size(0), -1) # flatten
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

B. Optimize Network Width

To identify the optimal architecture, we experimented with different values of *p* (neurons in the only layer). The tested configurations included:

- Hidden layers: **1**
- Neurons per layer: **2, 8, 32, 128**

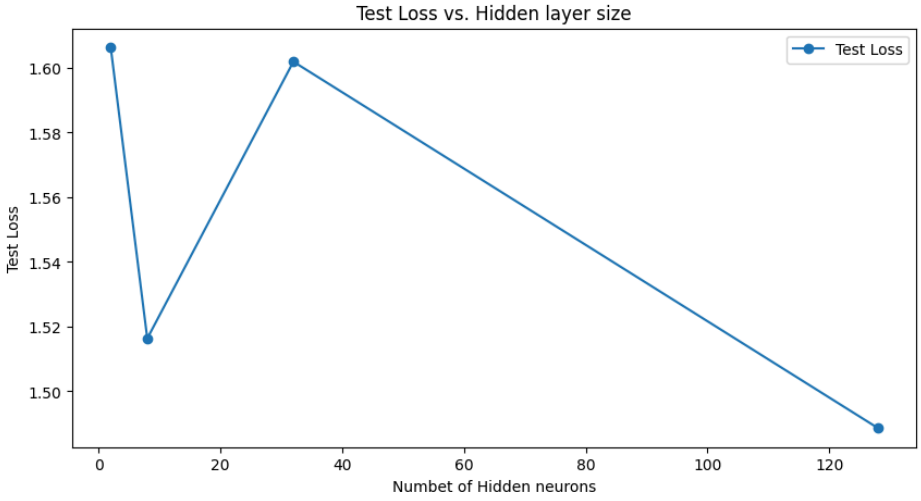
```
# various snippets of code in this block were autofilled in using AI

hidden_dims = [2,8,32,128]
test_losses = []
test_accuracies = []

input_dim = X_train.shape[1] # Input size from SMILES embeddings
output_dim = y_train.shape[1] # Number of drug classes

for hidden_dim in hidden_dims:
 print(f'Training model with hidden dimension {hidden_dim} num of
neurons...')
 model = ShallowNN(input_dim, hidden_dim, output_dim).to(device)
 criterion = nn.CrossEntropyLoss()
 optimizer = optim.Adam(model.parameters(), lr=0.001)

 train_model(model, train_loader, criterion, optimizer, epochs = 5)
 test_loss, test_accuracy = evaluate_model(model, test_loader,
criterion)
```

```
 test_losses.append(test_loss)
 test_accuracies.append(test_accuracy)
 print(f'Test Loss: {test_loss:.4f}, Test Accuracy:
{test_accuracy:.4f}')
```

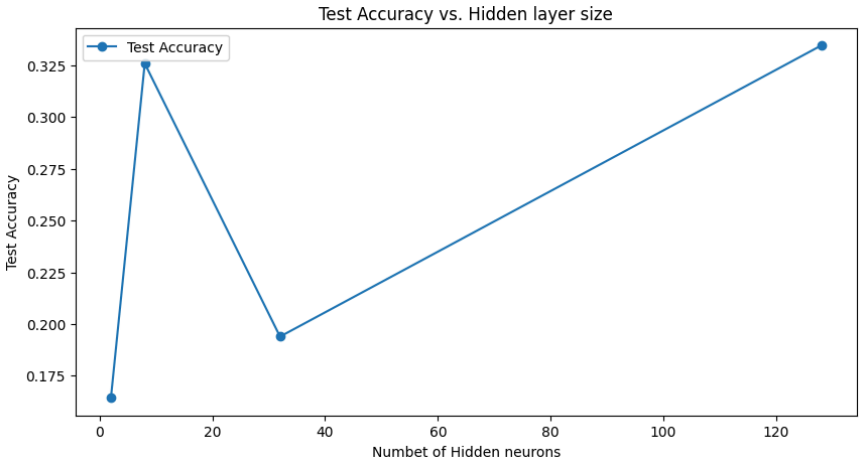
Each model was trained for **5 epochs** using a batch size of **256** to improve efficiency.

C. Look at Results in a Plot

This plot summarizes all model test loss vs. hidden layer size results with their respective hyperparameter tuning results.



This plot summarizes all model test accuracy vs. hidden layer size results with their respective hyperparameter tuning results.



Deep Network Analysis

D. Create a Deep Neural Network

We implemented a deep neural network with the following configurations:

- **Hidden layers: 1, 2, and 3 layers**
- **Neurons per layer: 32, 64, and 128**
- **Activation Function: ReLU**
- **Loss Function: CrossEntropyLoss**
- **Optimizer: Adam** (learning rate = **0.001**)
- **Batch Size: 256** (optimized for A100 GPU)
- **Training and Testing Dataset Shape:**
  - **X_train**: (5548, 399, 128)
  - **y_train**: (5548, 5)
  - **X_test**: (1387, 399, 128)
  - **y_test**: (1387, 5)

```
# defining a neural network (debugging assisted by OpenAI's ChatGPT)
class DeepNetwork(nn.Module):
  def __init__(self, input_size, hidden_layers, hidden_units,
output_size):
    super(DeepNetwork, self).__init__()
    layers = []
    layers.append(nn.Linear(input_size, hidden_units))
    layers.append(nn.ReLU())

    for _ in range(hidden_layers - 1):
      layers.append(nn.Linear(hidden_units, hidden_units))
      layers.append(nn.ReLU())

    layers.append(nn.Linear(hidden_units, output_size))
    self.model = nn.Sequential(*layers)

  def forward(self, x):
    x = x.view(x.size(0), -1) # flattens the output
    return self.model(x)
```

E. Optimize Network Depth and Width

To identify the optimal architecture, we experimented with different values of *d* (hidden layers) and *p* (neurons per layer). The tested configurations included:

- Hidden layers: **1, 2, 3**
- Neurons per layer: **32, 64, 128**

```
# testing optimization
batch_size = 256 # trying different batch sizes

# experimenting with depths and widths
hidden_layers_list = [1, 2, 3]
hidden_units_list = [32, 64, 128]

# looping through different model configurations to find the best one (large
portions of code are autofilled by google colab's gemini)
for hidden_layers in hidden_layers_list:
  for hidden_units in hidden_units_list:
    print(f"Training model with {hidden_layers} hidden layers and
{hidden_units} hidden units")
    deep_model = DeepNetwork(input_size = X_train.shape[1] * X_train.shape[2],
hidden_layers = hidden_layers, hidden_units = hidden_units, output_size =
y_train.shape[1]).to(device)
    train_losses, test_losses = train_deep_model(deep_model, train_loader,
test_loader, epochs = 5, lr = 0.001)

    final_test = test_losses[-1]
    results.append((hidden_layers, hidden_units, final_test))

    if final_test < best_loss:
      best_loss = final_test
      best_config = (hidden_layers, hidden_units)
```

Each model was trained for **5 epochs** using a batch size of **256** to improve efficiency.

4. Compare Results

The best and simplest performing deep network was **1 hidden layer with 64 neurons**, achieving a test loss of **1.302**. *2 hidden layers with 128 neurons* had a slightly better test loss but with an increased complexity, however, since we are going for the simplest model, 1 hidden layer with 64 neurons is best. The optimal hidden layer size for the small neural networks was achieved at **128 neurons in a single layer** with a test loss of **1.4887**.

Findings for the Shallow Network Analysis

- The increase of neurons in the only hidden layer appears to have contributed to an increase of model performance.
- These increases in performance are only true when using the accuracy and test loss as our main criteria.
- If other performance metrics were used, they may tell a different story.
- Perhaps some additional model performance increases would be present if we had the time to run each model for longer than 5 epochs.

Findings for the Deep Network Analysis

- Increasing the number of layers beyond 1 did not significantly improve performance.
- Adding more neurons beyond 64 per layer did not provide additional benefits, or was minimal at best. This suggests that the dataset does not require excessive depth.
- A single hidden layer with 64 neurons was sufficient to capture complex patterns in the data, preventing overfitting and reducing unnecessary computation.
- Any additional depth did not improve performance, likely because the dataset complexity does not require a deeper architecture. Therefore, the optimal balance of depth vs computational efficiency was found with 1 layer and 64 neurons.