

Lab 4: Implementing various Recurrent neural network cells using basic tensorflow ops

Nicole Sylvester

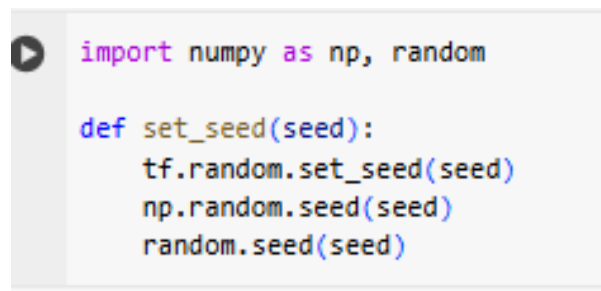
May 7, 2025
CS599 — Spring 2025

GitHub Link: <https://github.com/nicolesylvester/CS599Lab4>

1 Problem 1: Implementing various Recurrent neural network cells using basic tensorflow ops

1.1 Unique Seed

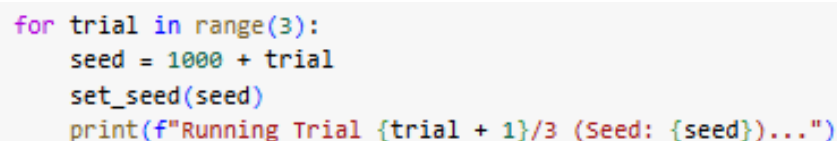
To ensure reproducibility, each model was trained across three trials using a consistent seeding strategy. **Seeds 1000, 1001, and 1002 were used for Trials 1, 2, and 3, respectively.** This allowed for fair comparisons across trials.

A code snippet in a light gray box with a play button icon on the left. The code defines a function set_seed(seed) that sets the seed for TensorFlow, NumPy, and the random module.

```
import numpy as np, random

def set_seed(seed):
    tf.random.set_seed(seed)
    np.random.seed(seed)
    random.seed(seed)
```

Figure 1: Set seed function

A code snippet in a light gray box showing a loop that runs three trials, each with a unique seed value (1000, 1001, 1002).

```
for trial in range(3):
    seed = 1000 + trial
    set_seed(seed)
    print(f"Running Trial {trial + 1}/3 (Seed: {seed})...")
```

Figure 2: Incrementing seed for trials

1.2 Model Implementation

The GRU and MGU cells were implemented by defining custom Keras layers and manually computing each gate using low-level TensorFlow operations like `tf.matmul`, `tf.sigmoid`, and `tf.tanh`. Both cells inherit from a shared `BaseRNNCell` class and define their own gating logic in the `call()` method. Each model was built by stacking one to four of these custom cells using `tf.keras.layers.RNN`, followed by a dense layer for classification. The `notMNIST_small` dataset was used, consisting of grayscale 28×28 images of letters A–J. Corrupted images were removed using PIL before loading the dataset with `image_dataset_from_directory`. The images were normalized to $[0, 1]$ and labels were one-hot encoded. A total of 80% of the data was used for training and 20% for validation. The model was compiled with the Adam optimizer (learning rate 0.001) and trained using categorical cross-entropy loss. Each model

configuration was trained for 20 epochs across three trials using random seeds 1000, 1001, and 1002 to ensure reproducibility. Configurations tested included four combinations of hidden units (128, 256, 512) and hidden layers (1, 2, 4) for both GRU and MGU models.

1.3 Results

Each model was trained over three trials per configuration. Final validation accuracy and classification error are reported below for both GRU and MGU models.

1.3.1 GRU Trials

- **GRU (1 layer, 128 units):**
 - Trial 1: Accuracy = 0.9215, Error = 0.0785
 - Trial 2: Accuracy = 0.9284, Error = 0.0716
 - Trial 3: Accuracy = 0.9346, Error = 0.0654
- **GRU (2 layers, 256 units):**
 - Trial 1: Accuracy = 0.9356, Error = 0.0644
 - Trial 2: Accuracy = 0.9354, Error = 0.0646
 - Trial 3: Accuracy = 0.9378, Error = 0.0622
- **GRU (4 layers, 128 units):**
 - Trial 1: Accuracy = 0.9340, Error = 0.0660
 - Trial 2: Accuracy = 0.9407, Error = 0.0593
 - Trial 3: Accuracy = 0.9322, Error = 0.0678
- **GRU (2 layers, 512 units):**
 - Trial 1: Accuracy = 0.9412, Error = 0.0588
 - Trial 2: Accuracy = 0.9356, Error = 0.0644
 - Trial 3: Accuracy = 0.9418, Error = 0.0582

1.3.2 MGU Trials

- **MGU (1 layer, 128 units):**
 - Trial 1: Accuracy = 0.9193, Error = 0.0807
 - Trial 2: Accuracy = 0.9185, Error = 0.0815
 - Trial 3: Accuracy = 0.9249, Error = 0.0751
- **MGU (2 layers, 256 units):**
 - Trial 1: Accuracy = 0.9340, Error = 0.0660
 - Trial 2: Accuracy = 0.9249, Error = 0.0751
 - Trial 3: Accuracy = 0.9316, Error = 0.0684
- **MGU (4 layers, 128 units):**
 - Trial 1: Accuracy = 0.9362, Error = 0.0638
 - Trial 2: Accuracy = 0.9308, Error = 0.0692
 - Trial 3: Accuracy = 0.9346, Error = 0.0654
- **MGU (2 layers, 512 units):**
 - Trial 1: Accuracy = 0.9343, Error = 0.0657
 - Trial 2: Accuracy = 0.9367, Error = 0.0633
 - Trial 3: Accuracy = 0.9390, Error = 0.0610

1.3.3 Accuracy Table

The table below summarizes the final validation accuracy of each model configuration over three independent trials. Each row lists the architecture (GRU or MGU), number of layers, hidden units, accuracy from all three trials, and the mean and standard deviation across those trials. GRU models consistently achieved slightly higher mean accuracy than MGU models across most configurations, with the GRU 2-layer 512-unit model achieving the highest mean accuracy overall.

	Model	Hidden Layers	Hidden Units	Trial 1	Trial 2	Trial 3	Mean Accuracy	Std Dev
1	GRU	1	128	0.9215	0.9284	0.9346	0.9282	0.0054
2	GRU	2	256	0.9356	0.9354	0.9378	0.9363	0.0011
3	GRU	4	128	0.9340	0.9407	0.9322	0.9356	0.0037
4	GRU	2	512	0.9412	0.9356	0.9418	0.9395	0.0028
5	MGU	1	128	0.9193	0.9185	0.9249	0.9209	0.0028
6	MGU	2	256	0.9340	0.9249	0.9316	0.9302	0.0039
7	MGU	4	128	0.9362	0.9308	0.9346	0.9339	0.0023
8	MGU	2	512	0.9343	0.9367	0.9399	0.9367	0.0019

Table 1: Accuracy Comparison

1.3.4 Accuracy Plot

The training and validation accuracy curves over 20 epochs are shown for each model configuration, averaged over three trials. GRU models learned faster and had slightly better validation accuracy during training. The shaded areas show how much the results varied between trials. For deeper models, both GRU and MGU became stable earlier, but GRU still ended with slightly better accuracy.

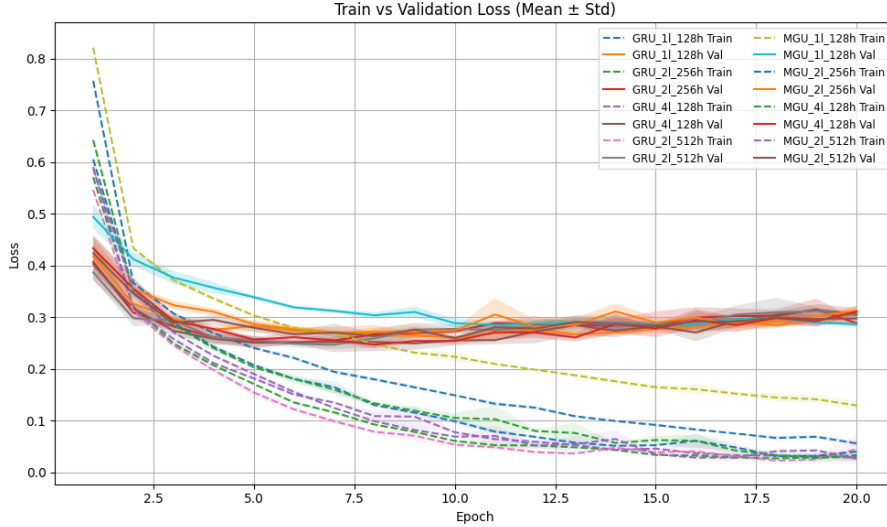


Figure 3: Train vs Validation Loss Plot

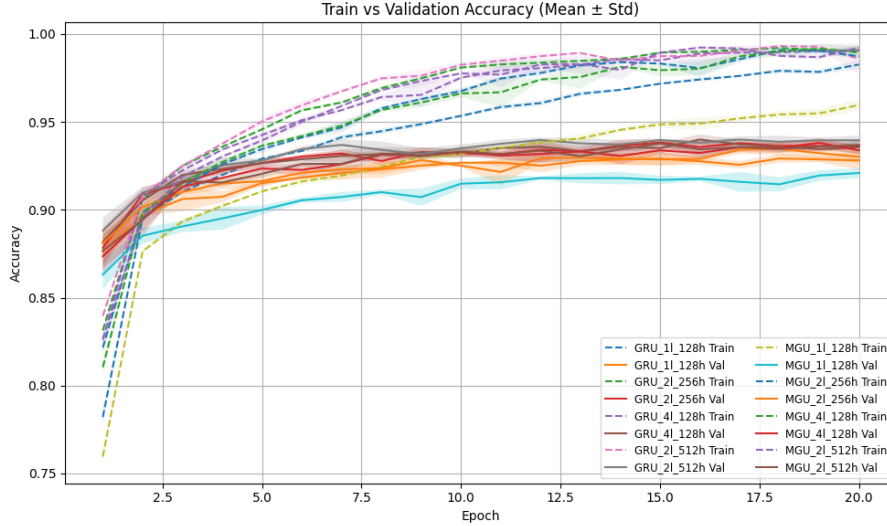


Figure 4: Train vs Validation Accuracy Plot

1.4 Discussion

1.4.1 GRU vs. MGU Comparison

GRU models performed better than MGU models in most cases, with higher average accuracy. The best GRU combination (2 layers, 512 units) reached around 94% accuracy, slightly better than the top MGU combination. GRU models were also more consistent, with less variation between trials, especially in deeper or larger models. MGU was still competitive and used fewer parameters but showed a bit more variation, especially with 2 layers and 256 units.

1.4.2 Impact of Hidden Units

Increasing hidden units improved validation accuracy for both GRU and MGU. GRU models benefited from larger hidden units, with the 512-unit configuration achieving the best overall performance. For MGU, the improvement from 128 to 512 units was smaller, which suggests that adding more units didn't help as much after a certain point.

1.4.3 Impact of Number of Hidden Layers

Adding more layers generally helped performance, especially for GRU models at 2 and 4 layers. MGU showed a similar pattern, with the 4-layer models doing about as well as the 2-layer ones. However, more layers didn't always lead to better results and sometimes made training less stable, likely because deeper networks are harder to optimize.

1.5 Conclusion

This experiment demonstrated that both GRU and MGU cells, when implemented using low-level TensorFlow operations, can effectively classify the notMNIST dataset. GRU consistently achieved higher validation accuracy and showed more stable performance across trials. MGU, while slightly less accurate on average, performed competitively with reduced complexity, making it a viable lightweight alternative. Increasing hidden size and number of layers generally improved performance, but deeper networks also introduced more variability. Overall, GRU offered the best trade-off between accuracy and consistency, while MGU provided efficiency with minimal loss in classification performance.