Contents

1 GPU Support in RNN UDFs

1.1 Overview

By default, the RNN model runs predictions in **CPU-only mode** within Spark executor UDFs to ensure maximum compatibility. However, if your cluster has GPUs available on worker nodes, you can enable GPU acceleration for inference by setting `use_gpu_in_udf=True`.

1.2 Quick Start

```
from skol_classifier.model import create_model

# Enable GPU in UDFs
model = create_model(
    model_type='rnn',
    input_size=2000,
    use_gpu_in_udf=True,  # Enable GPU acceleration
    verbosity=2
)
```

Or via SkolClassifierV2:

```
from skol_classifier.classifier_v2 import SkolClassifierV2

classifier = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=['data/*.txt'],
    model_type='rnn',
    use_gpu_in_udf=True,  # Enable GPU in prediction UDFs
    verbosity=2
)
```

1.3 When to Use GPU in UDFs

1.3.1 ✅ Use GPU in UDFs when:

- **All worker nodes have GPUs** with proper CUDA drivers
- **TensorFlow and CUDA versions are compatible** across all workers
- **GPU memory is sufficient** for your batch size (controlled by `prediction_batch_size`)
- **You need faster inference** on large datasets
- **You've tested GPU initialization** works in your Spark executors

1.3.2 ❌ Don't use GPU in UDFs when:

- Worker nodes don't have GPUs
- GPUs have incompatible CUDA versions
- You're running on a shared cluster with GPU contention
- GPU memory is limited
- You haven't verified GPU setup on workers

1.4 Default Behavior (CPU-Only Mode)

By default, `use_gpu_in_udf=False`, which means:

1. Sets `CUDA_VISIBLE_DEVICES=''` in executor processes
2. Calls `tf.config.set_visible_devices([], 'GPU')` to disable GPU
3. All inference runs on CPU

This ensures: - ✓ Maximum compatibility across different cluster configurations - ✓ No CUDA initialization errors - ✓ Works on CPU-only clusters - ✓ No GPU memory issues

1.5 GPU-Enabled Mode

When `use_gpu_in_udf=True`:

1. Does **not** set `CUDA_VISIBLE_DEVICES`
2. Allows TensorFlow to detect and use available GPUs
3. Enables memory growth on detected GPUs
4. Falls back to CPU if no GPU is available

This provides: - ✓ Faster inference with GPU acceleration - ✓ Automatic GPU detection per executor - ✓ Memory growth to prevent OOM errors - ✓ Graceful fallback to CPU if GPU unavailable

1.6 Requirements for GPU Mode

1.6.1 1. Hardware

- NVIDIA GPU on each worker node (or GPU-enabled executors)
- Compatible CUDA compute capability (check TensorFlow compatibility matrix)

1.6.2 2. Software

- CUDA toolkit installed on worker nodes
- cuDNN library installed
- TensorFlow GPU version (check with `python -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"`)

1.6.3 3. Spark Configuration

- Ensure executors have access to GPUs
- Configure executor GPU resource allocation if needed:

```
    spark = SparkSession.builder \
        .config("spark.executor.resource.gpu.amount", "1") \
        .config("spark.task.resource.gpu.amount", "1") \
        .getOrCreate()
```

1.7 Performance Considerations

1.7.1 Memory Usage

GPU memory usage depends on: - **Model size**: Larger models (more layers, bigger hidden size) use more memory - **Batch size**: Controlled by `prediction_batch_size` parameter - **Window size**: Larger windows increase memory usage

Recommended batch sizes: - Small models (1-2 layers, 64-128 hidden): `prediction_batch_size=128` - Medium models (2-3 layers, 128-256 hidden): `prediction_batch_size=64` - Large models (3+ layers, 256+ hidden): `prediction_batch_size=32`

1.7.2 Speed Improvement

GPU acceleration typically provides: - **2-5x speedup** for small models - **5-10x speedup** for medium models - **10-20x speedup** for large models

Actual speedup depends on: - GPU model and memory - Model architecture - Batch size - Input size and sequence length

1.8 Example: Training and Prediction with GPU

```
from pyspark.sql import SparkSession
from skol_classifier.classifier_v2 import SkolClassifierV2
import redis

# Create Spark session with GPU support
spark = SparkSession.builder \
    .appName("SKOL GPU Training") \
    .config("spark.executor.resource.gpu.amount", "1") \
    .getOrCreate()

# Initialize classifier with GPU enabled
classifier = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=['data/train/*.ann'],
    output_dest='files',
```

```
    output_path='data/output',
    model_type='rnn',

    # RNN model parameters
    input_size=2000,
    hidden_size=256,
    num_layers=3,
    window_size=50,
    prediction_batch_size=64,  # Adjust based on GPU memory
    epochs=20,

    # Enable GPU in UDFs
    use_gpu_in_udf=True,

    verbosity=2
)

# Train model (happens on driver or workers depending on setup)
print("Training model...")
classifier.fit_and_save_model()

# Predict with GPU acceleration
print("Making predictions with GPU...")
predictions = classifier.predict()

# Save results
classifier.save_annotated(predictions)
```

1.9 Troubleshooting

1.9.1 Problem: CUDA Initialization Error

Error message:

```
Failed to create CUDA stream: CUDA_ERROR_INVALID_HANDLE
```

Solutions: 1. Verify GPU compatibility: `nvidia-smi` on worker nodes 2. Check TensorFlow GPU support: `python -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"` 3. Ensure CUDA version matches TensorFlow requirements 4. Try disabling GPU: `use_gpu_in_udf=False`

1.9.2 Problem: Out of Memory Error

Error message:

```
Resource exhausted: OOM when allocating tensor
```

Solutions: 1. Reduce `prediction_batch_size` (e.g., from 64 to 32) 2. Reduce `window_size` if possible 3. Use smaller model (fewer layers or smaller hidden size) 4. Fall back to CPU: `use_gpu_in_udf=False`

1.9.3 Problem: GPU Not Detected

Symptoms: - UDF logs show "no GPUs detected - will use CPU" - Performance is same as CPU mode

Solutions: 1. Check GPU visibility: `nvidia-smi` on worker nodes 2. Verify CUDA_VISIBLE_DEVICES is not set globally 3. Check Spark executor logs for GPU assignment 4. Ensure executor has GPU resource allocated

1.9.4 Problem: Multiple Executors on Same GPU

Symptoms: - Inconsistent performance - Random CUDA errors - Memory errors

Solutions: 1. Configure Spark to assign GPUs to executors properly: `python .config("spark.task.resource.gpu.amount", "1") .config("spark.executor.resource.gpu.amount", "1")` 2. Limit executors per node to match GPU count 3. Use GPU isolation via CUDA_VISIBLE_DEVICES at executor level

1.10 Monitoring GPU Usage

1.10.1 During Training

Training happens on the driver (or master node), so monitor GPU on driver:

```
# On driver node
watch -n 1 nvidia-smi
```

1.10.2 During Prediction

Predictions run in executors, so monitor worker GPUs:

```
# On each worker node
watch -n 1 nvidia-smi
```

1.10.3 Logging GPU Status

Enable verbosity to see GPU detection in UDF logs:

```
classifier = SkolClassifierV2(
    # ... other params ...
    use_gpu_in_udf=True,
    verbosity=3  # Shows GPU detection messages
)
```

Check executor logs for messages like:

```
[UDF PROBA] TensorFlow imported, 1 GPU(s) available with memory growth enabled
```

1.11 Best Practices

1.11.1 1. Test Before Production

Always test GPU mode on a small dataset first:

```
# Test with small sample
test_classifier = SkolClassifierV2(
    spark=spark,
    file_paths=['data/test_sample/*.txt'],  # Small sample
    use_gpu_in_udf=True,
    verbosity=3  # Enable detailed logging
)
test_predictions = test_classifier.predict()
# Verify no CUDA errors before scaling up
```

1.11.2 2. Start Conservative

Begin with conservative settings and increase gradually:

```
# Start conservative
prediction_batch_size=32  # Small batch

# Monitor GPU memory usage
# Increase if memory allows
prediction_batch_size=64  # Medium batch
prediction_batch_size=128  # Large batch (if GPU has >8GB)
```

1.11.3 3. Use CPU Mode for Development

During development and testing, use CPU mode for consistency:

```
# Development
use_gpu_in_udf=False  # Consistent across all machines

# Production (after testing)
use_gpu_in_udf=True  # Enable for performance
```

1.11.4 4. Monitor Resource Usage

Track GPU utilization to ensure efficiency: - Target 70-90% GPU utilization - If <50%, increase batch size - If >95%, decrease batch size to avoid OOM

1.12 Comparison: CPU vs GPU Mode

Aspect	CPU Mode (Default)	GPU Mode
Compatibility	✓ Universal	Requires GPU setup
Setup Complexity	✓ Simple	Complex (CUDA, drivers)
Performance	Baseline	2-20x faster
Memory Usage	Low	Higher (GPU memory)
Failure Risk	Low	Medium (CUDA errors)
Cost	Lower (CPU only)	Higher (GPU instances)
Recommended For	Development, testing, CPU clusters	Production with large datasets

1.13 Example: Hybrid Deployment

You can use CPU for training (one-time) and GPU for inference (repeated):

```
# Training (one-time, can use CPU)
train_classifier = SkolClassifierV2(
    spark=spark,
    file_paths=['data/train/*.ann'],
    model_type='rnn',
    use_gpu_in_udf=False,  # CPU for training (runs on driver anyway)
    verbosity=2
)
train_classifier.fit_and_save_model()

# Inference (repeated, use GPU for speed)
pred_classifier = SkolClassifierV2(
    spark=spark,
    file_paths=['data/new_data/*.txt'],
    model_type='rnn',
    auto_load_model=True,
```

```
    use_gpu_in_udf=True,  # GPU for fast inference
    prediction_batch_size=128,
    verbosity=2
)
predictions = pred_classifier.predict()
```

1.14 See Also

- RNN Model Documentation
- SkolClassifierV2 API
- TensorFlow GPU Guide
- Spark GPU Configuration

Version: RNN Model 2.0+ **Updated**: 2025-12-20