FaileModel

June 7, 2025

[3]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, random_split
from sklearn.utils.class_weight import compute_class_weight
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt


WINDOW = 30
LABELS = [0, 1, 2, 3, 4]  # Wake, N1, N2, N3, REM
```

[4]:
```
df = pd.read_csv('/Users/veeralpatel/ECE284FinalProject/data/
 ↪S002_PSG_df_updated.csv', usecols=[
    'ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR', 'Sleep_Stage'
])

stage_map = {
    "W": 0,
    "N1": 1,
    "N2": 2,
    "N3": 3,
    "R": 4
}

df = df.copy()
df['Sleep_Stage'] = df['Sleep_Stage'].map(stage_map)
df = df.dropna(subset=['Sleep_Stage'])
df['Sleep_Stage'] = df['Sleep_Stage'].astype(int)
```

[5]:
```
WINDOW = 30  # 1 Hz sampling = 30 seconds
X_segments, y_segments = [], []
channels = ["ACC_X", "ACC_Y", "ACC_Z", "TEMP", "EDA", "HR"]

for i in range(0, len(df) - WINDOW, WINDOW):
    chunk = df.iloc[i:i+WINDOW]
```

```
    if chunk["Sleep_Stage"].nunique() == 1:  # single label per epoch
        X_segments.append(chunk[channels].values)
        y_segments.append(chunk["Sleep_Stage"].iloc[0])

X = np.stack(X_segments)
y = np.array(y_segments)
print(f"Total epochs: {len(X)} - Shape: {X.shape}")
```

```
Total epochs: 65312 - Shape: (65312, 30, 6)
```

[6]:
```
class SleepDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

dataset = SleepDataset(X, y)
train_len = int(0.8 * len(dataset))
train_set, val_set = random_split(dataset, [train_len, len(dataset) -␣
 ↪train_len])
train_loader = DataLoader(train_set, batch_size=32, shuffle=True)
val_loader = DataLoader(val_set, batch_size=32)


class CNN_BiLSTM_Model(nn.Module):
    def __init__(self, input_channels=6, num_classes=5):
        super(CNN_BiLSTM_Model, self).__init__()
        self.conv1 = nn.Conv1d(input_channels, 64, kernel_size=5, stride=1)
        self.bn1 = nn.BatchNorm1d(64)
        self.pool1 = nn.MaxPool1d(kernel_size=2)

        self.conv2 = nn.Conv1d(64, 128, kernel_size=3)
        self.bn2 = nn.BatchNorm1d(128)

        self.lstm = nn.LSTM(input_size=128, hidden_size=128,␣
 ↪bidirectional=True, batch_first=True)

        self.fc1 = nn.Linear(256, 64)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = x.permute(0, 2, 1)  # (batch, channels, time)
```

```
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = F.relu(self.bn2(self.conv2(x)))
        x = x.permute(0, 2, 1)  # (batch, time, features)
        x, _ = self.lstm(x)
        x = x[:, -1, :]
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

[9]:
```
weights = compute_class_weight(class_weight='balanced', classes=np.unique(y),␣
 ↪y=y)
class_weights = torch.tensor(weights, dtype=torch.float32)
criterion = nn.CrossEntropyLoss(weight=class_weights)

model = CNN_BiLSTM_Model(input_channels=6, num_classes=5)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

EPOCHS = 75
for epoch in range(EPOCHS):
    model.train()
    total_loss, correct, total = 0, 0, 0

    for xb, yb in train_loader:
        optimizer.zero_grad()
        out = model(xb)
        loss = criterion(out, yb)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        correct += (out.argmax(1) == yb).sum().item()
        total += yb.size(0)

    acc = correct / total
    print(f"Epoch {epoch+1}: Loss = {total_loss:.4f}, Train Acc = {acc:.4f}")
```

```
Epoch 1: Loss = 1013.8506, Train Acc = 0.7522
Epoch 2: Loss = 820.5557, Train Acc = 0.7934
Epoch 3: Loss = 744.4093, Train Acc = 0.8097
Epoch 4: Loss = 699.4798, Train Acc = 0.8197
Epoch 5: Loss = 667.8167, Train Acc = 0.8269
Epoch 6: Loss = 623.4382, Train Acc = 0.8362
Epoch 7: Loss = 606.9489, Train Acc = 0.8406
Epoch 8: Loss = 584.8319, Train Acc = 0.8461
Epoch 9: Loss = 560.1630, Train Acc = 0.8508
Epoch 10: Loss = 556.9440, Train Acc = 0.8520
Epoch 11: Loss = 542.3478, Train Acc = 0.8549
Epoch 12: Loss = 530.5164, Train Acc = 0.8571
```

```
Epoch 13: Loss = 511.8450, Train Acc = 0.8618
Epoch 14: Loss = 507.9238, Train Acc = 0.8650
Epoch 15: Loss = 498.1925, Train Acc = 0.8645
Epoch 16: Loss = 492.9588, Train Acc = 0.8673
Epoch 17: Loss = 479.3014, Train Acc = 0.8699
Epoch 18: Loss = 476.2592, Train Acc = 0.8708
Epoch 19: Loss = 470.2599, Train Acc = 0.8730
Epoch 20: Loss = 461.4718, Train Acc = 0.8751
Epoch 21: Loss = 456.2912, Train Acc = 0.8746
Epoch 22: Loss = 455.7859, Train Acc = 0.8774
Epoch 23: Loss = 448.1630, Train Acc = 0.8790
Epoch 24: Loss = 443.0459, Train Acc = 0.8811
Epoch 25: Loss = 438.3809, Train Acc = 0.8805
Epoch 26: Loss = 436.3698, Train Acc = 0.8818
Epoch 27: Loss = 433.4494, Train Acc = 0.8816
Epoch 28: Loss = 432.5599, Train Acc = 0.8832
Epoch 29: Loss = 421.3876, Train Acc = 0.8852
Epoch 30: Loss = 422.7147, Train Acc = 0.8866
Epoch 31: Loss = 411.3866, Train Acc = 0.8882
Epoch 32: Loss = 408.6607, Train Acc = 0.8886
Epoch 33: Loss = 413.8441, Train Acc = 0.8880
Epoch 34: Loss = 408.6330, Train Acc = 0.8886
Epoch 35: Loss = 406.1438, Train Acc = 0.8885
Epoch 36: Loss = 411.4846, Train Acc = 0.8882
Epoch 37: Loss = 398.3122, Train Acc = 0.8912
Epoch 38: Loss = 400.2597, Train Acc = 0.8913
Epoch 39: Loss = 391.2328, Train Acc = 0.8927
Epoch 40: Loss = 393.9160, Train Acc = 0.8922
Epoch 41: Loss = 385.8131, Train Acc = 0.8935
Epoch 42: Loss = 390.0074, Train Acc = 0.8941
Epoch 43: Loss = 387.2803, Train Acc = 0.8947
Epoch 44: Loss = 386.3065, Train Acc = 0.8944
Epoch 45: Loss = 382.4812, Train Acc = 0.8945
Epoch 46: Loss = 385.3838, Train Acc = 0.8945
Epoch 47: Loss = 380.0021, Train Acc = 0.8963
Epoch 48: Loss = 379.8832, Train Acc = 0.8977
Epoch 49: Loss = 374.1201, Train Acc = 0.8968
Epoch 50: Loss = 373.5412, Train Acc = 0.8969
Epoch 51: Loss = 372.4660, Train Acc = 0.8984
Epoch 52: Loss = 374.0214, Train Acc = 0.8991
Epoch 53: Loss = 369.1882, Train Acc = 0.8991
Epoch 54: Loss = 367.5717, Train Acc = 0.8991
Epoch 55: Loss = 365.8802, Train Acc = 0.9001
Epoch 56: Loss = 367.8725, Train Acc = 0.8995
Epoch 57: Loss = 360.3513, Train Acc = 0.9011
Epoch 58: Loss = 362.4597, Train Acc = 0.9013
Epoch 59: Loss = 363.2964, Train Acc = 0.9004
Epoch 60: Loss = 359.8788, Train Acc = 0.9010
```

```
Epoch 61: Loss = 357.7483, Train Acc = 0.9021
Epoch 62: Loss = 352.6868, Train Acc = 0.9033
Epoch 63: Loss = 354.4657, Train Acc = 0.9030
Epoch 64: Loss = 353.6179, Train Acc = 0.9028
Epoch 65: Loss = 358.7299, Train Acc = 0.9021
Epoch 66: Loss = 352.6774, Train Acc = 0.9030
Epoch 67: Loss = 351.9976, Train Acc = 0.9038
Epoch 68: Loss = 347.0274, Train Acc = 0.9044
Epoch 69: Loss = 346.2135, Train Acc = 0.9059
Epoch 70: Loss = 349.1489, Train Acc = 0.9040
Epoch 71: Loss = 344.6163, Train Acc = 0.9061
Epoch 72: Loss = 349.2698, Train Acc = 0.9037
Epoch 73: Loss = 347.6872, Train Acc = 0.9054
Epoch 74: Loss = 344.4418, Train Acc = 0.9060
Epoch 75: Loss = 343.8008, Train Acc = 0.9066
```

[10]:
```python
model.eval()
correct, total = 0, 0
all_preds, all_labels = [], []

with torch.no_grad():
    for xb, yb in val_loader:
        out = model(xb)
        preds = out.argmax(1)
        all_preds.extend(preds.tolist())
        all_labels.extend(yb.tolist())
        correct += (preds == yb).sum().item()
        total += yb.size(0)

print(f"  Validation Accuracy: {correct/total:.4f}")

from sklearn.metrics import classification_report
print(classification_report(all_labels, all_preds, zero_division=0))
torch.save(model.state_dict(), "dreamt.pt")
```

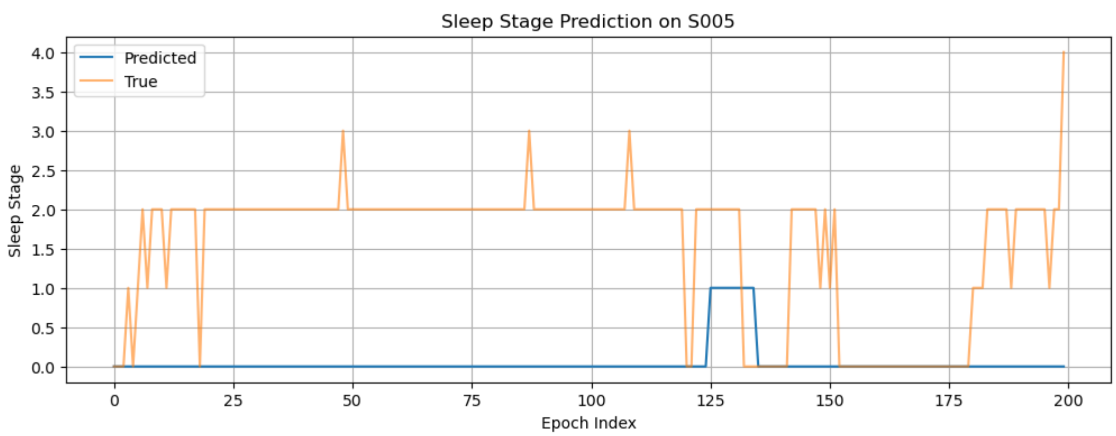
```
 Validation Accuracy: 0.9171
              precision    recall  f1-score   support

           0       0.90      0.91      0.91      4451
           1       0.78      0.77      0.78      1032
           2       0.95      0.94      0.95      6578
           4       0.88      0.93      0.90      1002

    accuracy                           0.92     13063
   macro avg       0.88      0.89      0.88     13063
weighted avg       0.92      0.92      0.92     13063
```

[15]:
```
df = pd.read_csv('/Users/veeralpatel/ECE284FinalProject/data/
 ↪S005_PSG_df_updated.csv', usecols=[
    'ACC_X', 'ACC_Y', 'ACC_Z', 'TEMP', 'EDA', 'HR', 'Sleep_Stage'
])

stage_map = {
    "W": 0,
    "N1": 1,
    "N2": 2,
    "N3": 3,
    "R": 4
}

df = df.copy()
df['Sleep_Stage'] = df['Sleep_Stage'].map(stage_map)
df = df.dropna(subset=['Sleep_Stage'])
df['Sleep_Stage'] = df['Sleep_Stage'].astype(int)

# === 2. Extract 30s epochs ===
WINDOW = 3000  # assuming 100Hz * 30s
features = ["ACC_X", "ACC_Y", "ACC_Z", "TEMP", "EDA", "HR"]

X_segments, y_segments = [], []
print("Unique labels in Sleep_Stage:", df["Sleep_Stage"].unique())
print("DataFrame length:", len(df))

for i in range(0, len(df) - WINDOW, WINDOW):
    chunk = df.iloc[i:i + WINDOW]
    mode_label = chunk["Sleep_Stage"].mode().iloc[0]
    feat = chunk[features].values
    X_segments.append(feat)
    y_segments.append(mode_label)

X_s005 = np.stack(X_segments)
y_s005 = np.array(y_segments)

X_s005_tensor = torch.tensor(X_s005, dtype=torch.float32)
y_s005_tensor = torch.tensor(y_s005, dtype=torch.long)

# === 3. Reshape: (batch, time, features) ===
X_s005_tensor = X_s005_tensor.permute(0, 1, 2)  # already (N, T, C)

# === 4. Load model ===
model.load_state_dict(torch.load("dreamt.pt"))
model.eval()

# === 5. Predict ===
```

```python
with torch.no_grad():
    y_pred = model(X_s005_tensor).argmax(dim=1)

# === 6. Evaluation ===
print("  Classification Report on S005:")
print(classification_report(y_s005_tensor, y_pred, target_names=["W", "N1",␣
 ↪"N2", "N3", "R"]))

# === 7. Optional visualization ===
plt.figure(figsize=(12, 4))
plt.plot(y_pred[:200], label="Predicted")
plt.plot(y_s005_tensor[:200], label="True", alpha=0.6)
plt.title("Sleep Stage Prediction on S005")
plt.ylabel("Sleep Stage")
plt.xlabel("Epoch Index")
plt.legend()
plt.grid(True)
plt.show()
```

```
Unique labels in Sleep_Stage: [0 1 2 3 4]
DataFrame length: 2198997

/var/folders/jd/lt8pv90x74741r8x6g2_d5lh0000gn/T/ipykernel_73376/4026704073.py:4
3: FutureWarning: You are using `torch.load` with `weights_only=False` (the
current default value), which uses the default pickle module implicitly. It is
possible to construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  model.load_state_dict(torch.load("dreamt.pt"))

 Classification Report on S005:
              precision    recall  f1-score   support

           W       0.13      0.97      0.22        93
          N1       0.00      0.00      0.00        39
          N2       0.00      0.00      0.00       479
          N3       0.00      0.00      0.00         5
           R       0.00      0.00      0.00       116

    accuracy                           0.12       732
   macro avg       0.03      0.19      0.04       732
```

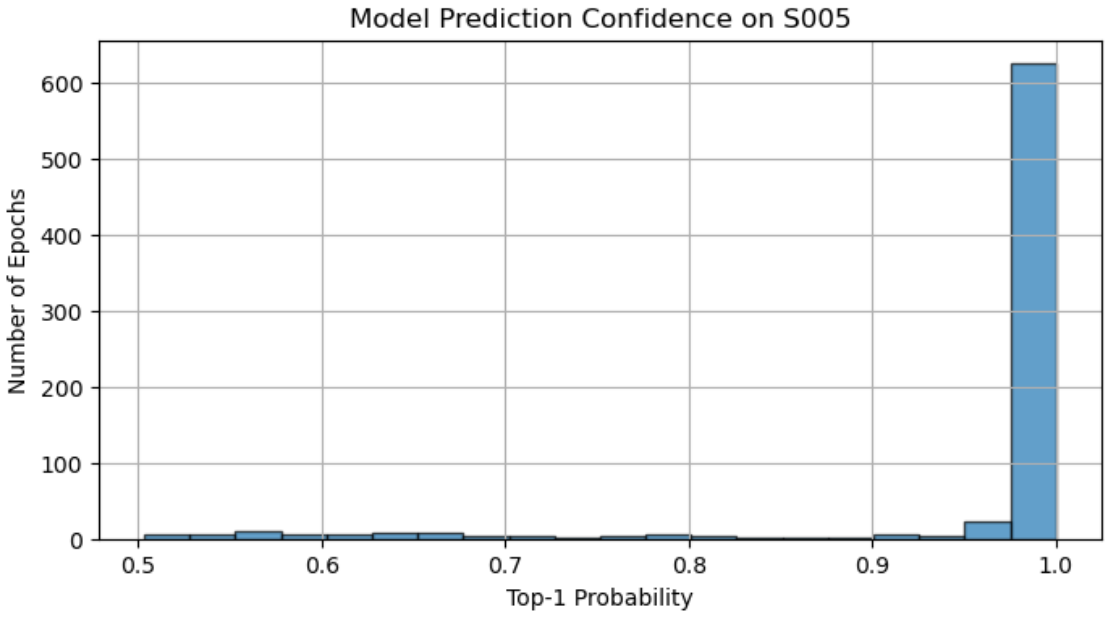
```
weighted avg       0.02      0.12      0.03       732

/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



[16]:
```python
with torch.no_grad():
    logits = model(X_s005_tensor)
    probs = torch.softmax(logits, dim=1)
    preds = probs.argmax(dim=1)

# === 2. Show sample predictions and confidence ===
top_probs = probs.max(dim=1).values
print("  Sample predicted stage labels:", preds[:10].tolist())
print("  Sample confidences:", top_probs[:10].tolist())
print("  Avg confidence across all predictions:", round(top_probs.mean().
 ↪item(), 3))

# === 3. Plot histogram of predicted probabilities ===
```

```
plt.figure(figsize=(8, 4))
plt.hist(top_probs.numpy(), bins=20, alpha=0.7, edgecolor='black')
plt.title("Model Prediction Confidence on S005")
plt.xlabel("Top-1 Probability")
plt.ylabel("Number of Epochs")
plt.grid(True)
plt.show()

# === 4. Breakdown of predictions by class ===
unique, counts = np.unique(preds.numpy(), return_counts=True)
print("  Prediction distribution (stage: count):")
for u, c in zip(unique, counts):
    print(f"  Stage {u}: {c} predictions")

# === 5. Optional: Show confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_s005_tensor, preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["W", "N1",␣
 ↪"N2", "N3", "R"])
disp.plot(cmap="Blues")
plt.title("Confusion Matrix - S005")
plt.show()
```

```
  Sample predicted stage labels: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  Sample confidences: [0.9998660087585449, 0.999150276184082,
0.9989925026893616, 0.998053789138794, 0.9931029081344604, 0.9873084425926208,
0.9905852675437927, 0.993939995765686, 0.9940263032913208, 0.9960338473320007]
  Avg confidence across all predictions: 0.962
```



```
Prediction distribution (stage: count):
 Stage 0: 718 predictions
 Stage 1: 14 predictions
```