Spike Inference

Using a Bi-Directional LSTM to predict spikes from LFP data

- Leonardo Ferrisi
- Alana Maluszczak
- Daniel Feldman


```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Bidirectional, LSTM, Dropout, Dense,
LayerNormalization
from keras.optimizers import Adam
from keras.regularizers import l2
import time

def create_sequences(signal, labels, window_size):
    """
    Create sequences from the 1D time series signal.
    For each window of LFP data, the label is the spike value at the
time immediately after the window.
    """
    X, y = [], []
    for i in range(len(signal) - window_size):
        X.append(signal[i : i + window_size])
        y.append(labels[i + window_size])
    return np.array(X), np.array(y)

# ------------------------------
# 1. Data Loading and Preprocessing
# ------------------------------
# Change the data_dir and filename as needed.
data_dir = "./data"
filename = "fake_lfp_data.csv"  # CSV file generated previously
data_path = os.path.join(data_dir, filename)

if not os.path.exists(data_path):
    print(f"Data file {data_path} not found. Please ensure the CSV
exists.")
    raise Exception("Data file not found")

# Read the CSV file; expecting columns: 'time', 'lfp', and 'spike'
```

```
df = pd.read_csv(data_path)

# Extract the LFP and spike columns (assuming LFP is our feature and
spike is our label)
lfp = df['lfp'].values
spike = df['spike'].values

# -------------------------------
# 2. Creating Sequences for the LSTM
# -------------------------------
# Define a window size (number of timesteps per sample)
window_size = 100  # You can adjust this based on your sampling rate &
desired context

# Create sequences using the sliding window approach.
# The label for each sequence is the spike value immediately after the
window.
X, y = create_sequences(lfp, spike, window_size)

# Reshape X to have shape (samples, timesteps, features). In this
case, features=1.
X = X.reshape(-1, window_size, 1)

print("Data shapes:")
print("X:", X.shape)
print("y:", y.shape)

Data shapes:
X: (9900, 100, 1)
y: (9900,)

# -------------------------------
# 3. Splitting the Dataset: 70% Training, 30% Validation
# -------------------------------
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"Training set shape: X_train={X_train.shape},
y_train={y_train.shape}")
print(f"Validation set shape: X_val={X_val.shape},
y_val={y_val.shape}")

Training set shape: X_train=(6930, 100, 1), y_train=(6930,)
Validation set shape: X_val=(2970, 100, 1), y_val=(2970,)

# -------------------------------
# 4. Building the Bidirectional LSTM Model
# -------------------------------
input_timesteps = X_train.shape[1]
input_features = X_train.shape[2]
```

```
model = Sequential([
    # First Bidirectional LSTM layer; return_sequences=True to allow
stacking
    Bidirectional(LSTM(64, return_sequences=True),
input_shape=(input_timesteps, input_features)),
    Dropout(0.2),

    # Second Bidirectional LSTM layer; return_sequences=False as it's
the last LSTM layer
    Bidirectional(LSTM(32, return_sequences=False)),
    Dropout(0.2),

    # Final Dense layer for binary classification (predicting spike or
no spike)
    Dense(1, activation='sigmoid')
])


time_start_model = time.time()

model.compile(loss='binary_crossentropy',
optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
model.summary()

time_end_model = time.time()
print(f"Model building time: {time_end_model - time_start_model}
seconds")
```

```
Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━
━━━━━┩
│ bidirectional_2 (Bidirectional) │ (None, 100, 128)       │
33,792 │
├─────────────────────────────────┼────────────────────────┼─────────
─────┤
│ dropout_2 (Dropout)             │ (None, 100, 128)       │
0 │
├─────────────────────────────────┼────────────────────────┼─────────
─────┤
│ bidirectional_3 (Bidirectional) │ (None, 64)             │
41,216 │
├─────────────────────────────────┼────────────────────────┼─────────
─────┤
│ dropout_3 (Dropout)             │ (None, 64)             │
```

```
0 │
├──────────────────────────────────┼─────────────────────────┼───────
──────┤
│ dense_1 (Dense)                  │ (None, 1)               │
65 │
└──────────────────────────────────┴─────────────────────────┴───────
──────┘

 Total params: 75,073 (293.25 KB)

 Trainable params: 75,073 (293.25 KB)

 Non-trainable params: 0 (0.00 B)

Model building time: 0.014992952346801758 seconds

# ------------------------------
# 5. Training the Model
# ------------------------------

time_start_training = time.time()

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,         # Adjust the number of epochs as needed
    batch_size=64,     # Adjust batch size as needed
    verbose=1
)

time_end_training = time.time()
print(f"Model training time: {time_end_training - time_start_training}
seconds")

Epoch 1/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 17s 99ms/step - accuracy: 0.8812 - loss:
0.3597 - val_accuracy: 0.9212 - val_loss: 0.2057
Epoch 2/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 92ms/step - accuracy: 0.9261 - loss:
0.1928 - val_accuracy: 0.9152 - val_loss: 0.1942
Epoch 3/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9242 - loss:
0.1829 - val_accuracy: 0.9229 - val_loss: 0.1769
Epoch 4/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 94ms/step - accuracy: 0.9255 - loss:
0.1770 - val_accuracy: 0.9229 - val_loss: 0.1863
Epoch 5/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - accuracy: 0.9281 - loss:
0.1690 - val_accuracy: 0.9246 - val_loss: 0.1724
Epoch 6/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - accuracy: 0.9285 - loss:
```

```
0.1644 - val_accuracy: 0.9256 - val_loss: 0.1673
Epoch 7/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 94ms/step - accuracy: 0.9285 - loss:
0.1672 - val_accuracy: 0.9253 - val_loss: 0.1701
Epoch 8/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 11s 96ms/step - accuracy: 0.9260 - loss:
0.1676 - val_accuracy: 0.9276 - val_loss: 0.1666
Epoch 9/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 94ms/step - accuracy: 0.9295 - loss:
0.1689 - val_accuracy: 0.9266 - val_loss: 0.1727
Epoch 10/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - accuracy: 0.9321 - loss:
0.1566 - val_accuracy: 0.9256 - val_loss: 0.1687
Epoch 11/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9304 - loss:
0.1605 - val_accuracy: 0.9236 - val_loss: 0.1708
Epoch 12/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 92ms/step - accuracy: 0.9256 - loss:
0.1709 - val_accuracy: 0.9253 - val_loss: 0.1681
Epoch 13/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 94ms/step - accuracy: 0.9285 - loss:
0.1618 - val_accuracy: 0.9259 - val_loss: 0.1688
Epoch 14/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - accuracy: 0.9328 - loss:
0.1613 - val_accuracy: 0.9239 - val_loss: 0.1694
Epoch 15/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - accuracy: 0.9312 - loss:
0.1615 - val_accuracy: 0.9263 - val_loss: 0.1676
Epoch 16/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 92ms/step - accuracy: 0.9299 - loss:
0.1674 - val_accuracy: 0.9229 - val_loss: 0.1724
Epoch 17/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9289 - loss:
0.1598 - val_accuracy: 0.9246 - val_loss: 0.1701
Epoch 18/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9272 - loss:
0.1571 - val_accuracy: 0.9263 - val_loss: 0.1675
Epoch 19/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 96ms/step - accuracy: 0.9304 - loss:
0.1640 - val_accuracy: 0.9269 - val_loss: 0.1678
Epoch 20/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9325 - loss:
0.1585 - val_accuracy: 0.9242 - val_loss: 0.1738
Epoch 21/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9255 - loss:
0.1641 - val_accuracy: 0.9266 - val_loss: 0.1691
Epoch 22/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9243 - loss:
0.1641 - val_accuracy: 0.9269 - val_loss: 0.1658
```

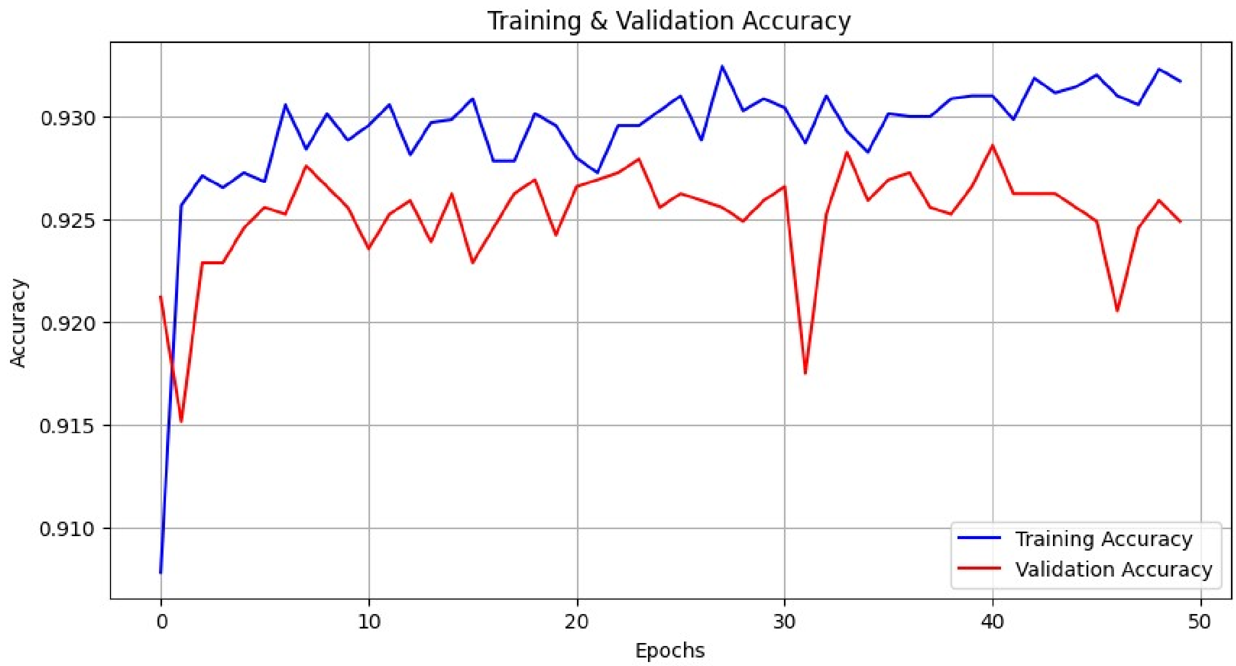
```
Epoch 23/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9323 - loss:
0.1599 - val_accuracy: 0.9273 - val_loss: 0.1652
Epoch 24/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9325 - loss:
0.1545 - val_accuracy: 0.9279 - val_loss: 0.1684
Epoch 25/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9323 - loss:
0.1581 - val_accuracy: 0.9256 - val_loss: 0.1658
Epoch 26/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9245 - loss:
0.1648 - val_accuracy: 0.9263 - val_loss: 0.1658
Epoch 27/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9309 - loss:
0.1591 - val_accuracy: 0.9259 - val_loss: 0.1635
Epoch 28/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 92ms/step - accuracy: 0.9330 - loss:
0.1542 - val_accuracy: 0.9256 - val_loss: 0.1645
Epoch 29/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 90ms/step - accuracy: 0.9322 - loss:
0.1605 - val_accuracy: 0.9249 - val_loss: 0.1692
Epoch 30/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9328 - loss:
0.1536 - val_accuracy: 0.9259 - val_loss: 0.1658
Epoch 31/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9245 - loss:
0.1664 - val_accuracy: 0.9266 - val_loss: 0.1686
Epoch 32/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9295 - loss:
0.1547 - val_accuracy: 0.9175 - val_loss: 0.1741
Epoch 33/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 90ms/step - accuracy: 0.9322 - loss:
0.1553 - val_accuracy: 0.9253 - val_loss: 0.1683
Epoch 34/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9289 - loss:
0.1579 - val_accuracy: 0.9283 - val_loss: 0.1645
Epoch 35/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9280 - loss:
0.1605 - val_accuracy: 0.9259 - val_loss: 0.1661
Epoch 36/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - accuracy: 0.9250 - loss:
0.1671 - val_accuracy: 0.9269 - val_loss: 0.1663
Epoch 37/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - accuracy: 0.9275 - loss:
0.1601 - val_accuracy: 0.9273 - val_loss: 0.1718
Epoch 38/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 94ms/step - accuracy: 0.9304 - loss:
0.1574 - val_accuracy: 0.9256 - val_loss: 0.1689
Epoch 39/50
```

```
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 88ms/step - accuracy: 0.9324 - loss:
0.1593 - val_accuracy: 0.9253 - val_loss: 0.1677
Epoch 40/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9300 - loss:
0.1520 - val_accuracy: 0.9266 - val_loss: 0.1654
Epoch 41/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9236 - loss:
0.1664 - val_accuracy: 0.9286 - val_loss: 0.1649
Epoch 42/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 88ms/step - accuracy: 0.9287 - loss:
0.1585 - val_accuracy: 0.9263 - val_loss: 0.1679
Epoch 43/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9325 - loss:
0.1551 - val_accuracy: 0.9263 - val_loss: 0.1674
Epoch 44/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 87ms/step - accuracy: 0.9335 - loss:
0.1524 - val_accuracy: 0.9263 - val_loss: 0.1688
Epoch 45/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 88ms/step - accuracy: 0.9306 - loss:
0.1578 - val_accuracy: 0.9256 - val_loss: 0.1667
Epoch 46/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.9338 - loss:
0.1511 - val_accuracy: 0.9249 - val_loss: 0.1672
Epoch 47/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9273 - loss:
0.1579 - val_accuracy: 0.9205 - val_loss: 0.1729
Epoch 48/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.9258 - loss:
0.1643 - val_accuracy: 0.9246 - val_loss: 0.1694
Epoch 49/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9337 - loss:
0.1495 - val_accuracy: 0.9259 - val_loss: 0.1670
Epoch 50/50
109/109 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9283 - loss:
0.1570 - val_accuracy: 0.9249 - val_loss: 0.1677
Model training time: 506.54667472839355 seconds

# ------------------------------
# 6. Plotting Training History
# ------------------------------
# Plot Training & Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss',
color='red')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()
```

```
plt.grid()
plt.show()
```


```
# Plot Training & Validation Accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy',
color='blue')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy',
color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
plt.legend()
plt.grid()
plt.show()
```


Training & Validation Accuracy
Accuracy
0.930
0.925
0.920
0.915
0.910
0
10
20
30
40
50
Epochs
Training Accuracy
Validation Accuracy