Project Report: LSTM Model for Sequence Prediction

By Kamala Natarajan

Data Processing Steps:

The input sequences were tokenized using TensorFlow's Tokenizer with char_level=True to handle each character in the sequences separately. This approach was chosen because each character in the sequence can represent specific biological properties when predicting molecular or biological outcomes from sequences. Post tokenization, sequences were padded to a uniform length of 500 using pad_sequences. This maximum length was determined through exploratory data analysis, identifying that it covers the length of most sequences without losing much information on the shorter sequences. Explicit data cleaning was not necessary as the sequences were assumed to be pre-validated and error-free, typical in controlled datasets for biological sequence prediction tasks. The primary feature engineering step involved transforming character sequences into numerical representations that could be fed into the neural network. This transformation was achieved through tokenization. No external data was included in this model to keep the focus on the provided dataset and ensure the integrity of the experiment's-controlled environment.

ML Model:

The Embedding layer converts tokenized integers into dense vectors of size 64. It helps in interpreting the input sequences by learning an embedding for all the characters in the dataset. Bidirectional LSTM layer contains 64 units, returns sequences to allow the next LSTM layer to receive sequences of outputs, enhancing the learning of patterns from both forward and reverse directions. The dropout Layer (0.5) was used after the LSTM layers to prevent overfitting by randomly setting input units to 0 at each update during training time.LSTM Layer was introduced containing 32 units, processing the output from the bidirectional LSTM. An additional dropout layer (0.5) was added to further mitigate the risk of overfitting. A dense Layer of 64 units with ReLU activation, adds non-linearity to the network, helping to learn complex patterns. A single neuron with a linear activation function to predict the target value as a continuous output was used as an output layer.

Rationale for Choice:

The bidirectional LSTM allows the model to capture dependencies in both directions of the sequence, which is crucial for understanding the context in sequence data that could have bidirectional influences. Dropout layers are critical in complex models like this to combat overfitting, especially when dealing with relatively small datasets in deep learning contexts.

ML Model Training:

The dataset was split into training and validation sets with a ratio of 80:20 using TensorFlow's built-in validation split functionality during training. The model was trained for 20 epochs with a batch size of 32. These parameters were chosen based on experimental tuning, balancing training time, and convergence performance. Adam optimizer was used due to its effectiveness in handling sparse gradients and adaptive learning rate capabilities, which are beneficial for deep learning models on sequence data.

Hyperparameter Selection:

The default learning rate of the Adam optimizer was used. Preliminary experiments with different rates showed minor improvements, thus the default rate was kept for simplicity. Tested with 16, 32, and 64. 32 provided the best trade-off between training speed and memory utilization. Chosen after observing the validation loss plateau around 20 epochs, indicating diminishing returns on further training. 0.5 was empirically chosen after testing values between 0.3 and 0.6, where 0.5 offered the best performance on the validation set without leading to excessive underfitting.

Results:

The model's performance was primarily validated using mean squared error on the validation set. Observations showed a gradual decrease in loss, confirming the model's ability to learn and generalize from the training data effectively. Besides the Kaggle leaderboard metrics, local validation using the mean squared error helped in fine-tuning and provided quick feedback on model adjustments.

Discussion:

The model performed adequately but showed signs of overfitting, as indicated by the smaller gap between training and validation loss. This aspect was mitigated to some extent using dropout layers. The cross-validation strategy was rigorous, but a k-fold cross-validation could potentially provide more robust validation, especially with different data splits. Initial versions of the model without dropout layers tended to overfit, seen by excellent training loss but poor validation loss. Adding dropout layers helped balance this. Altering the sequence length to 500 characters significantly enhanced prediction accuracy, striking a balance between capturing sufficient contextual data and maintaining computational efficiency. This length was chosen as it minimized information loss while preventing the model from becoming overwhelmed with noise, thus optimizing predictive performance.